Architecture of optimized digital twins for AI-based training

[WP2; T2.3; Deliverable: D2.3; Version]

Public

AI training using Simulated Instruments for Machine Optimization and Verification

Version	Status	Date	Page
Version 2.0	public	2024.05.08	1/69

Document Information

Project	ASIMOV
Grant Agreement No.	20216 ASIMOV - ITEA
Deliverable No.	D2.3
Deliverable No. in WP	WP2; T2.3
Deliverable Title	Architecture of optimized digital twins for AI-based training
Dissemination Level	external
Document Version	2.0
Date	2024.05.08
Contact	Richard Doornbos
Organization	TNO
E-Mail	Richard.Doornbos@tno.nl

The ASIMOV-project was submitted in the Eureka Cluster AI Call 2021
https://eureka-clusters-ai.eu/

Task Team (Contributors to this deliverable)

Name	Partner	E-Mail
Niklas Braun	AVL	Niklas.Braun@avl.com
Mehrnoush Hajnorouzi	DLR	Mehrnoush.Hajnorouzi@dlr.de
Elias Modrakowski	DLR	Elias.Modrakowski@dlr.de
Andreas Eich	LiangDao	Andreas.eich@liangdao.de
Narges Javaheri	TFS	Narges.Javaheri@thermofisher.com
Richard Doornbos	TNO	Richard.Doornbos@tno.nl
Sebastian Moritz	TrianGraphics	Sebastian.Moritz@triangraphics.de
Jan-Willem Bikker	CQM	JanWillem.Bikker@cqm.nl
Rutger van Beek	CQM	Rutger. Vanbeek@cqm.nl
Roy van Zuijlen	TU/e	r.a.c.zuijlen@tue.nl

Formal Reviewers

Version	Date	Reviewer
1.8	2024	Jan van Doremalen (CQM)
1.8	2024	Jilles van Hulst (TUE)

Change History

Version	Date	Reason for Change
0.7	2023.06.08	Content review version.
0.8	2023.06.28	Adaptation after content review.
0.9	2023.06.28	Formal review version.
1.0	2023.07.13	Intermediate version. (M24)
1.8	2024.03.28	Content review version.
2.0	2024.05.08	Final version. (M36)

Abstract

The ASIMOV project investigates the combination of Digital Twins (DTs) and Artificial Intelligence (AI) to find the opportunities and challenges for automated optimization and calibration of complex high-tech systems in complex environments. In many cases the actual system is not available for training AI components, therefore a dedicated digital twin or digital model is set up for providing that training data. The AI component is set up to capture the complexity of the real system and control and optimize the system in its operation.

This report focusses on the DT part and starts by describing the main challenges in creating a model mimicking the behavior of a real system. In the state-of-the-art section, the high-level architectural descriptions currently published are discussed.

This report describes next the reference architecture for a digital twin specifically for training AI systems. This reference architecture considers from various viewpoints how such a digital twin / digital model can be designed and developed. It should be used as *a guideline* for creating the specific architecture for a concrete digital twin / digital model. It contains generic elements, aspects, and best practices. This reference architecture description also provides challenges, concerns, and questions to be answered by the systems architects and system designers.

This architectural part is followed by extensive elaboration on modeling system behavior. It provides insights on which types of modeling exist, how to model behavior, and the consequences for use in AI training.

Contents

Table of Figures

Table of Tables

1 Introduction

The Asimov project investigates the combination of Digital Twins (DTs) and Artificial Intelligence (AI) to find the opportunities and challenges for automated optimization and calibration of complex high-tech systems in complex environments. This document describes specifically for these purposes the *reference architecture* for the digital twin. It should be used as a guideline for creating the architecture of a specific digital twin. It describes generic elements, discusses various aspects, and best practices. It also provides challenges, concerns, and questions which need to be answered by the systems architects. Furthermore, the last part is dedicated to DT behavior focusing on modeling, describing which types of modeling exist, and discussing the consequences for use in AI training.

The general role of a reference architecture in a company is shown in Figure 1. This simplified picture shows that a reference architecture guides the development of several product architectures and captures the commonalities of various (existing) products. It ensures that the company vision and strategy are executed and embedded in the architectures of the product portfolio. Other applications of reference architectures may have different purposes and different scopes: from sub-system level (e.g., system-internal communication infrastructure), to inter-company (e.g., Google's cloud architecture framework), or even domain reference architectures (e.g., the German Reference Architecture Model Automotive).

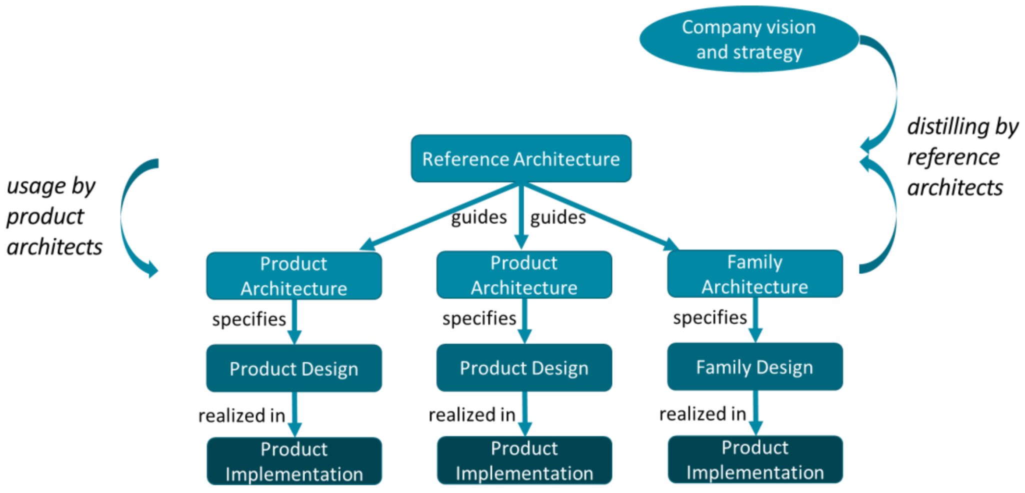

Figure 1 The role and creation of a reference architecture (in the context of a company).

To create a reference architecture, we attempt to extract the commonalities (and detect differences) in existing products and future systems. In the ASIMOV project, we consider the use cases (which are real implementations of the DT+AI solution) as 'products' for the distilling purposes. The vision and strategy will play a lesser role; however, this may be replaced by other driving forces.

1.1 Definitions

In this section some important concepts necessary for further reading and essential for understanding are introduced.

A *digital twin* (abbreviated to DT) is a digital replica of a physical entity with the two-way dynamic mapping between a physical object and its digital model [1]. Another definition is: a realistic digital representation of assets, processes, or systems in the built or natural environment [2]. The variation in definitions of the term digital twin is as large as the variation in digital twins themselves. Therefore, we will not dwell on the discussion about definitions and focus on digital twin characteristics, purposes, and structures for the Asimov context [3-7].

For simplicity, we adopt the definitions suggested by Aheleroff [3] that distinguish the types based on the synchronization of the digital twin ('Digital Replica') with the actual system ('Physical Thing'); see Figure 2.

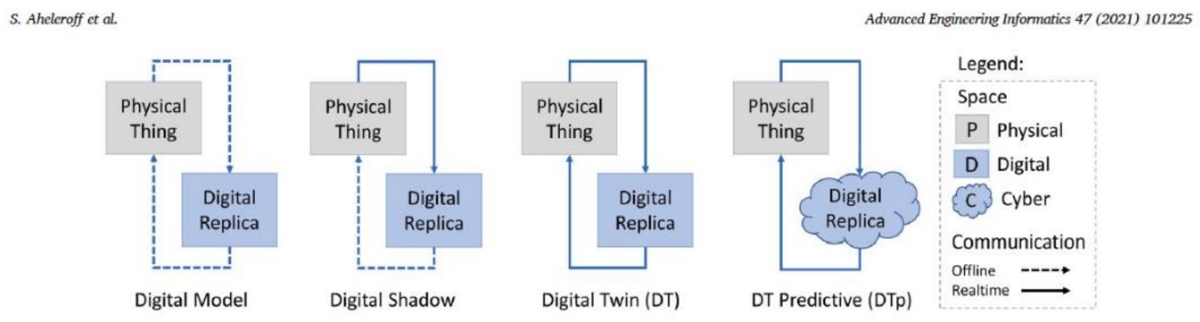

Figure 2 Definitions of digital twins (reproduced from *[3]*).

Digital twins have a set of core characteristics. Erikstad [4] identified five core characteristics: identity (connecting to a single, real and unique physical asset), representation (capturing of essential physical manifestation), state (rendering quantifiable measures of the asset´s state), behavior (reflecting basic responses to external stimuli), and context (external operating context in which the asset exists). These characteristics will be explored for the ASIMOV use cases and subsequently generalized for the digital twin in the role for training artificial intelligence components.

The benefits of a digital twin may be exploited especially in the case of training AI components: improved availability and control, high speed training, reduced cost of operation, possibility of investigating extreme or dangerous scenarios, etc.

The downside is that the digital twin requires substantial development investment and must be validated thoroughly and continuously to trust the results. The impact of incorrect behavior on the AI during training may be profound and only discovered late, even only in the operational phase. Note that this trade-off occurs in many places in R&D: an early investment is required to prevent later problems. The difficulty is to prove the investment is beneficial to convince company management.

Another concept is a *phase*. These have early on been defined as a stage in the development of the ASIMOV solution in the Working Group deliverable [5]. Here a summary of these concepts is reproduced:

Training phase: In this phase, the AI, or more specifically the RL agent (RLA), is trained using multiple varying model instances (DTs) of a typical cyber physical system (CPS). This is done to prevent the problem of overfitting and increase generalization to enable policy transfer.
Operational phase: Once the RLA has matured enough, it is deployed on an actual CPS (the physical twin) and performs system optimizations. In this phase, the RLA is assumed to interact directly with the CPS or is part of it. During this period data from the CPS is gathered for the next phase.
Fine-tuning phase: the RLA is Iteratively fine-tuned using its CPS's data from the operational phase and/or its DT to improve the RLA's accuracy of fit to the CPS instance, but also to further train the variant agnostic RLA (see result of the training phase), e.g., using federated learning. The frequency of this fine-tuning depends on the use case. A sequential but iterative cycle between operational and fine-tuning phase is recommended since adjustments to the RLA should be extensively tested before release, but - if online learning is implemented – the operational and fine-tuning phase may become one.

Another notion is the *Conceptual System Level*, which is important for an architect to understand the scope of stakeholder needs. In Figure 3, a useful structure of scopes is depicted by explicitly showing the 'nesting' of systems that are identified for the ASIMOV solution. These conceptual system levels help to pinpoint at which level particular concerns play a role. These concerns must all be addressed in the overall system architecture.

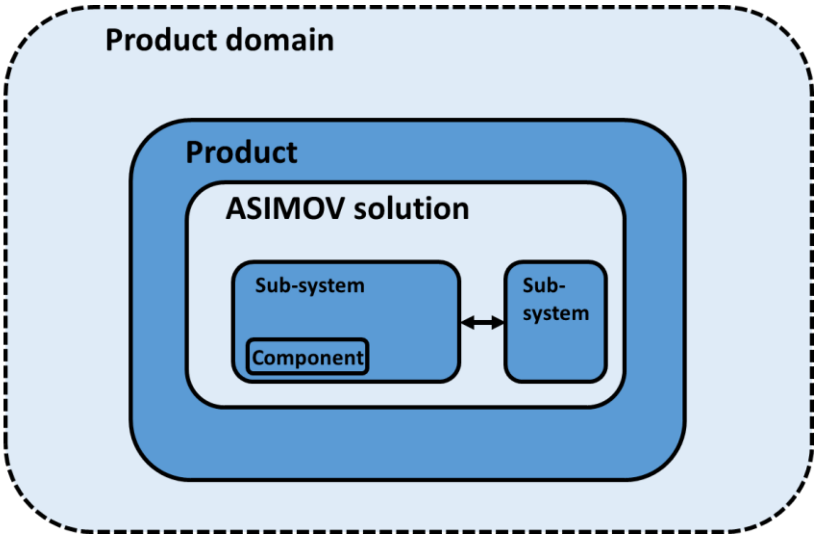

Figure 3 Conceptual system levels.

We distinguish five conceptual levels:

- **Product domain:** the context (systems or people) that interact with the product. This is the level for describing usage scenarios, customer concerns, business concerns, etc.
- **Product:** the system that will contain the AI-based optimization system.
- **ASIMOV solution:** the system-of-interest, shown here as a part of the product. The qualities of this solution (performance, scalability, etc.) are determined by its context (Product) and its constituents (Sub-systems).
- **Sub-system:** the main parts of the solution, primarily the DT and the AI.
- **Component:** the functional parts of a sub-system (e.g., controller, interface, storage).

One should be aware of the various phases in the development of systems. This holds for green-field development (starting from scratch), as well as for brownfield (continued development of an existing system or product). We can generally distinguish the following phases, each typically requiring 3 to 5 times more effort:

1) Proof-of-concept

Focus is on the feasibility of technologies: can it work? How to make it work?

2) Prototype

Focus is on the feasibility in the product context: will it work in our product? Can it work in the customer's environment?

3) Product

Focus is on the productization: how to professionalize the prototype solution? How to ensure business benefits?

In the various phases of development, different purposes lead to different concerns and requirements. We need to understand that in this document about the DT architectures the discussions apply to this variety in the development phases.

1.2 Challenges

Developing a DT for training an AI has many challenges. They reside at and across(!) various levels (see Figure 3), and occur in different phases of the development.

In this document we focus on the ‘DT for AI’ topic. What are the challenges that go beyond the ‘normal’ challenges [6] due to tensions in requirements and designs of systems and architecture solutions? What is typically more difficult in the DT-AI combination?

In the first section we distinguish the challenges in the development and integration of a DT into a bigger (AI) system. The second section describes the challenges arising in modelling the behavior of the physical system and its environment.

1.2.1 Architectural challenges

In **Proof-of-concept** development the typical questions and tensions arise: how feasible is the basic DT solution? Can we achieve the required qualities (e.g., speed and accuracy)? What is now specific due to the ‘AI-as-client’ of the DT?

An architect should consider the following challenging aspects in particular:

- The DT architecture development also involves the integration of many models from many different disciplines, which is a challenge on its own.
- The training data generated by the DT should be ‘rich’ in such a way that the training of the AI component is fast. Finding this richness is a formidable challenge, and very case specific.

In **Prototype** development the focus is on the integration in the AI (and possibly product) context. The challenges are in finding out how to control/manage the DT qualities (e.g., data reproducibility, data generation speed, response time), and in flexibility (changing the purpose of the DT: including different behaviors over time, ease of evolution of the future product). What are the architectural tensions in the chosen DT design?

An architect should consider the following challenging aspects in particular:

- The controllability (or tunability) of the digital twin output is a challenge on its own: especially the accuracy of the result. E.g., when far from the optimum, the accuracy does not have to be high, when being close to the optimum the accuracy becomes very important. The calculation time is coupled to the accuracy, so there is a trade-off between speed and accuracy.
- Flexibility: how to quickly adapt the DT for a different objective? In principle, the overall objective or purpose of the DT-AI combination determines the construction of both the DT (type of models, what is being modelled) and the AI (target, type of algorithms). This flow down of architectural drivers should be made explicit for each case. Also, the variability in product behaviors and configurations should be handled in this phase.

Product development has a different type of challenge, rooted in the final goal of creating business (i.e., generating money), compared to the previous phases. Topics such as productization, maintainability, deployment, and footprint come to the foreground.

An architect should consider the following challenging engineering aspects in particular:

- The DT is a company-internal product, and so in principle it should be developed, produced and maintained in a professional way, as if it were a real product. In practice, the status of the DT depends on the mindset, professionalism, and maturity of the company.
- DT is new technology for many people in a company. It may not be a part of the culture to do modeling of the entire system in its context and use a DT as the Digital Thread as a single source of truth.
- The field of digital twinning is fast developing. A high-tech equipment company has typically difficulty in keeping up, so special effort should be invested in this challenge. Reusability of DT (components) may be a real problem that requires attention.

More discussion on qualities and tensions can be found in Chapter 3.1.

1.2.2 Modelling challenges - a selection

Within the wider activities of developing a DT, twins or models of smaller components can be developed that are modelled with a more limited view. Such models may be entirely built from first principles or may in part rely on example data from a system. Generally, formulating a model or DT from first principles and domain knowledge without having to rely on data at all is in principle preferred over data-driven modelling. If data is involved, different challenges can arise as explained in this section.
Models are developed in a wide range of disciplines, with wide varieties of challenges. Writing down an equation from first principles or physics can be seen as an ideal situation. Also, a model can take on the form of a simulation model in software (e.g., a FEM, finite element method, with a detailed 3D representation of an object and laws of physics). A pure physics-based simulation might be deterministic. However, randomness can be an important element of simulations, for instance in the simulation of the electron microscope has inherently random physics behaviour, or a logistic case may have inherently random demand. The model can also be based on data, e.g., calibration of unknown model parameters, a.k.a. configuration parameters, unknown constants, or weights. Machine learning and statistics can be used to make a model based on example data. The way that data is collected may be observational, where some system runs its course, and the resulting quantities are recorded. In contrast, an experiment can be carried out, where randomization is applied as in clinical trials or in “Design of Experiments”, a branch of statistics often applied in industrial statistics. In the latter, cause-and-effect relations can be drawn because of randomization. For observational data, causal relations can also be described if a causal model is applied (see Judea Pearl’s work, [7] and [8]).

Data can take on many forms. Complex examples are text, audio, and images. Simpler examples are list of properties, e.g., a vector with scalar values for weight, speed, direction, and acceleration. An intermediate example could be a time series (force over time, position over time). An important consideration is whether to summarize the data. Certain models can take complex data as input (convolutional neural nets are specialized to take in images as input). Models in statistics and many in machine learning take in a fixed vector of quantities (scalar, categorical, or mixes). See ASIMOV D2.2 on post-processing and state shaping.

Challenge: summarizing, aggregation
In data-driven model building, summarizing data can be a major activity that determines the model quality. In summarizing data information may be lost, but modelling becomes easier, and the art is in finding the balance. One aspect is the aggregation level (e.g., a 2-D image to a few scalars for contrast, brightness; time series into one value; will a set of time series be aggregated in one overall number or one per time series). Another is the nature of the aggregation, e.g., mean and median, or their counterpart's median, inter quartile range that are robust for outliers. In machine learning, this summarizing is called *feature engineering.* As a side note, deep learning may be an alternative that accepts un-aggregated data; downsides are that large amounts of data are needed, and the resulting model is a black box.

Challenge: specifying variation
In ASIMOV, it is foreseen that variation is added to a DT; see ASIMOV D2.2. This may consist of drawing random numbers and applying them to DT inputs or parameters as additional noise. However, in more complex structures it can be challenging to describe randomness as the additional variation on a DT. E.g., random additions to some aspect may have to be made for multiple points in time and space. Suppose a digital twin of some CPS tracks temperature over time on many locations. Two temperature values that are close in time and space might need highly correlated random additions as large difference in distortions temperature between two nearby related points are typically similar. This needs careful mathematical modelling of how to combine independent random building blocks. Simple examples are a Brownian motion and autocorrelation in time series. A complex example is if a DT itself has a random multivariate component that is in part given by a correlation matrix. Introducing variations on the correlations in the matrix is a mathematical challenge, because the required mathematical properties are easily broken if noise is naively added to the matrix elements.

Challenge: quantifying effects/understanding model structure
In optimization and training AI, model predictions are of primary interest. However, in many situations it is desirable to understand the form (or *shape*) of a model $output = f(input, time)$ beyond using it as a

black box. It can be very useful to investigate the influence of the inputs, as some inputs might have little influence, or one input has more influence as soon as a second input takes on a certain value, etc. Navigating this structure with few inputs can be done graphically, but gets difficult from dimension say 5 and higher: in typical 2-D static plots, each dimension needs a plot element such as horizontal axis, panel rows, panel columns, symbol/line colours, and symbols; 3-D or interactive plots may not bring much more; see e.g. [9] and [10].

Challenge: prediction performance
Prediction performance (mean squared error, sensitivity/recall, etc.) is sometimes based on a small test set. There are also different schemes of forming test sets, such as (k-fold) cross validation. It can be challenging to pick a scheme. Small test sets result in statistical uncertainty of the prediction performance metric which is not always recognized. See [11].

1.2.3 Model adaptation in twinning

A basic view of the DT is that it mimics the input-output relation of a system or system's component. However, ideally the ASIMOV solution has the capability to adapt the DT to varying circumstances of its environment or changes in the system over time (e.g., wear, degradation, different environment, swapping parts, …). There are several possible views of model adaptation, e.g., from control theory, software architectural (section 3.3.3.3), and from data-based modelling (section 4.4.4 gives some strategies), where each view may give some challenges.

An illustration that model adaptation can be a challenge is as follows. Suppose a CPS with a DT is being controlled using the ASIMOV solution. An RL agent gives actions to the CPS to keep it on the desirable set point. Suppose that the input-output relation changes as in Figure 4. If the agent does a good job of keeping things steady, over time the input/output coordinates stay in the same region (the blue oval). In this case, a change in relation cannot be noticed.

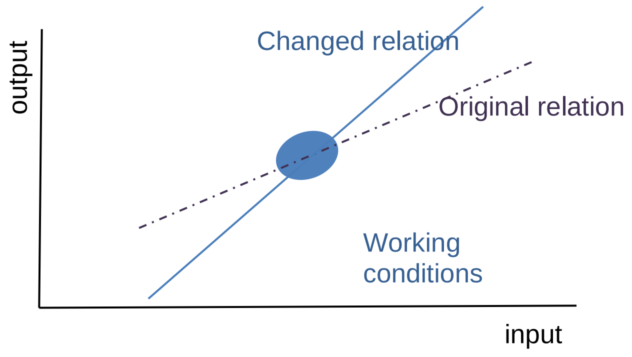

Figure 4 changing relation but with input/output staying in the same region over time

Some challenges
- How to set up an adaptation model in the architecture?
- How to deal with structure such as changes over time, or changes between variants in the system?
- How to detect changes in the relation; in a passive way, or even active (exploration/exploitation)?
- How to integrate an adaptation model in the basic DT approach?

2 State of the art on DT architectures and views

2.1 Existing architectural view frameworks

Like how problems can be seen by stakeholders, architectures can be seen from different viewpoints which highlight different aspects of the system. Being clear about what view is talked about eases the development of insight. In the following, a selection of literature on architectural views and viewpoints is shown to create a basic understanding.

2.1.1 Early architecture view model

One of the early works on architecture views is Kruchten's view model "4+1" [8]. It is "a model for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views" [8]. Five views are proposed:

- **Logical view:** The author defines the logical architecture to "primarily support[s] the functional requirements – what the system should provide in terms of services to its users" by decomposing the system into objects or object classes [8]. Kruchten proposes to use class diagrams for this matter to show "a set of classes and their logical relationships: association, usage, composition, inheritance, and so forth".
- **Process view:** The process architecture is concerned with non-functional requirements (performance, availability, etc.). It also shows the distribution of the system and aspects such as "system's integrity, fault-tolerance, and how the main abstractions from the logical view fit within the process architecture" by [8]. The author also states that this view can be seen in different levels of abstraction. With the process architecture, tasks and processes (groups of tasks) can be defined which is its main goal.
- **Development view:** The development view has the task of organizing the actual software modules into smaller chunks. "The development architecture of the system is represented by module and subsystem diagrams, showing the `export´ and `import´ relationships" [8]. It may be organized in layers with detailed interfaces in between them.
- **Physical view:** The physical architecture maps the software to the hardware. The previously identified elements such as objects, tasks and processes are mapped onto nodes which then represent hardware (module, computer, cluster of computers, etc.). Kruchten points out that "The mapping of the software to the nodes therefore needs to be highly flexible and have a minimal impact on the source code itself" [8].
- **Scenario view:** The scenario view is generally a use case or scenario related walkthrough through the system "describing the corresponding scripts (sequences of interactions between objects, and between processes)" [8]. The author points out that this view is the "+1" in the model's name "4+1 View Model" as it might not be necessary for sufficiently describing the system.

2.1.2 Reference architecture model of "Industrie 4.0"

A more modern standard for a reference architecture is the Reference Architectural Model Industrie 4.0, (RAMI 4.0) which describes all elements of industry 4.0 in a structured manner as presented in [3]. Digital Twins are a core component of the Industry 4.0 and can be described by RAMI 4.0. As shown in Figure 5, it consists of three axes:

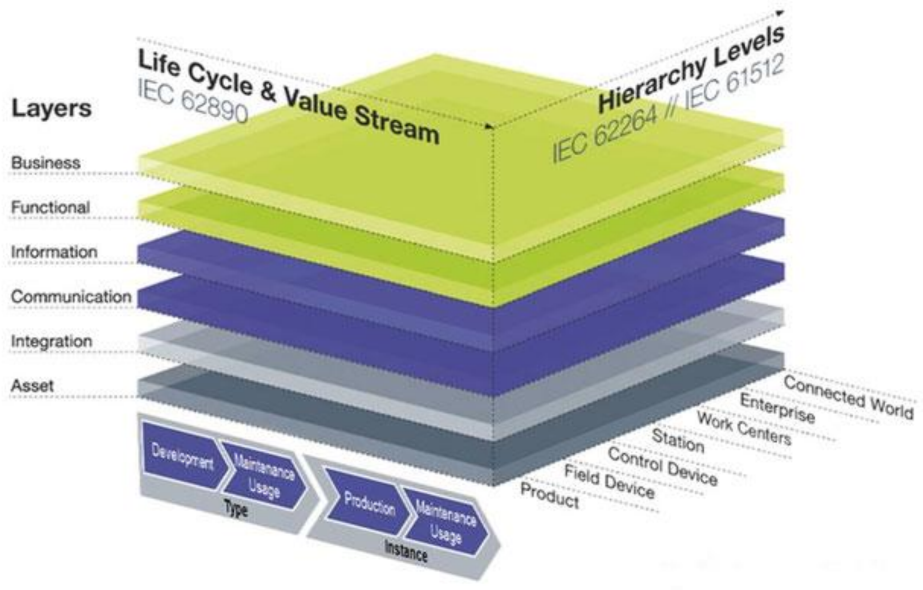

Figure 5 Visualization of the RAMI 4.0 reference architecture [3]

- **Hierarchy Levels:** On the right horizontal axis are the hierarchical levels from IEC 62264, the international series of norms on the integration of company IT and control systems. These hierarchy levels represent the different functionalities within the factory or plant. [3]
- **Life Cycle & Value Stream:** The left horizontal axis represents the life cycle of plants and products. The basis for this is IEC 62890 on life cycle management. A distinction is also made between type and instance. A "type" becomes an "instance" when the development and prototype production is completed, and the actual product is manufactured in the production department. [3]
- **Layers:** While the previous axis defines the level of detail and the point of time in the life cycle, the “layer” axis represents actual architectural viewpoints. With the help of the six layers on the vertical axis of the model, the IT representation, i.e., the digital image of a machine, for example, is described in a structured way layer by layer. The representation in layers comes from information and communication technology. In this field it is common practice to structure complex products in layers. [3]
    - **Business layer:** Orchestrating functions to form business processing and linking between different business processes. Includes high level requirements
    - **Functional layer:** Describes (logical) functions of an asset (technical functionality)
    - **Information layer:** Describes the data that is offered, used, generated or modified by the functions
    - **Communication layer:** Provides standard communication for service and event/data to the information layer, and services and control commands to the integration layer.
    - **Integration layer:** Describes the interface between physical components and the digital world
    - **Asset layer:** Represents the physical world such as physical components, software, documents and human actors.

2.1.3 Comparison of other architecture view models

The work of Brankovic et. al [9] is interesting and highly relevant as it combines several view model and reference architectures together and compares the proposed views as it can be seen Figure 6. These frameworks are namely Smart Grid Architecture Model (SGAM), Automotive Reference Architecture Model (ARAM) and the RAMI 4.0. They are compared to the more generalized Software Platform Embedded System (SPES) Framework, an approach designed for System of Systems-architectures.
The SPES defining document [12] explains the four viewpoints (VP):

- **Requirements VP:** This VP supports the engineering of requirements. It contains documentation of context, goals, specifications, and scenarios.
- **Function VP:** This VP provides the functional system specification based on the requirements which are determined in the previous VP. In other words, it defines the functionality the system is intended to provide.
- **Logical VP:** This VP supports the solution design, providing functional building blocks by communicating components arranged in a component architecture. It is a description of the platform-independent solution by subsystem decomposition
- **Technical VP:** This VP supports the technical implementation of the logical VP. It contains the description of the Hardware, Software tasks, scheduler, communication, Platform specific, etc., required to implement the system’s specification from the above VPs.

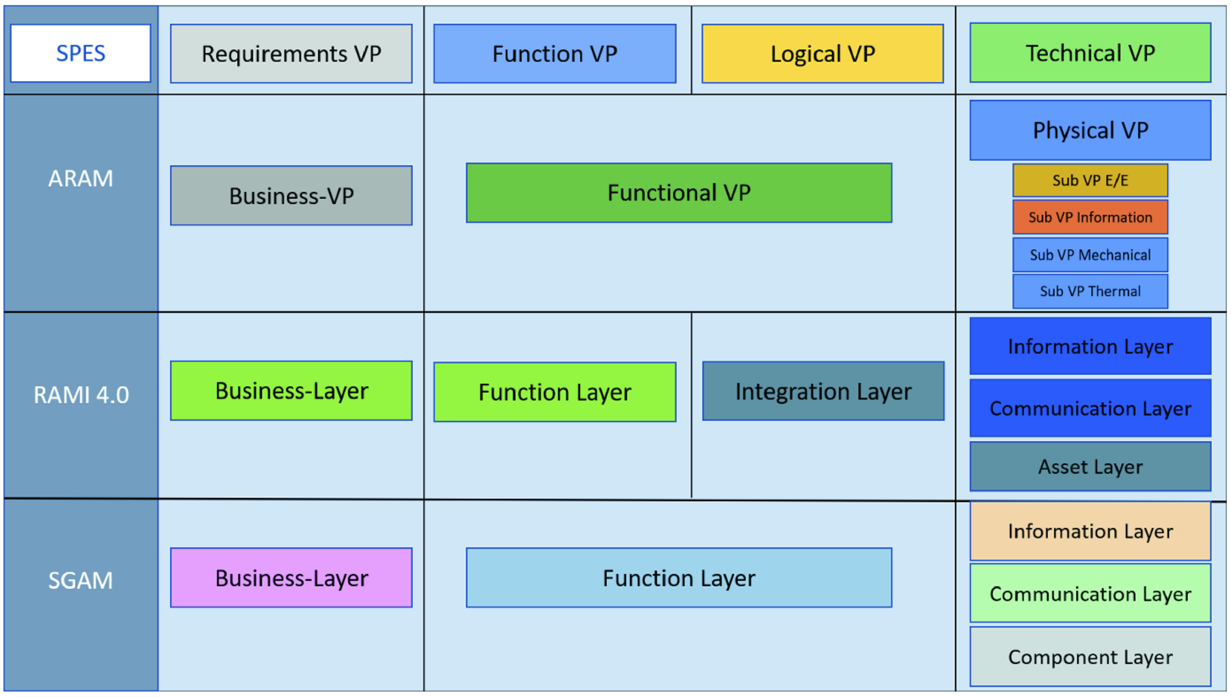

Figure 6 Mapping of different viewpoints and layers to SPES [9]

2.1.4 Summary and impact on DT architecture in ASIMOV

The work of Brankovic, et al. [9] shows that while there is a multitude of approaches to architectural VP, each focusing on different aspects in more detail, they all look at the system from similar angles. Most frameworks provide a VP for requirements on the system's components, their function and interconnectivity, and some technical specifications.
In general. SPES provides four general purpose VPs which seem to be a good trade-off between level of detail and adequacy for describing a reference architecture which shall be use-case/implementation agnostic. All other frameworks provide multiple additional VPs which are highly dependent on the platform; something we do not pursue to concretize in this work such as low-level technical specifications (APIs, ports,...). Thus, the aggregation to a technical VP provides the flexibility to point out solution independent specifications without being technology restrictive.

Thus, in the following section 3 "Architectural views on the DT for AI-training", we will look at the reference architecture from four VPs according to the SPES framework: Requirements, Function, Logical and Technical VP.

2.2 State of the art on Digital Twin architecture

A literature review on architectures of DTs of [13] identified 140 primary studies on this topic. Starting from 2016, the number of publications is growing exponentially. Most of these papers in [13] (86%) are solution oriented i.e., provide an architecture for specific purpose in a determined context. Even though the concept of Digital Twins is not new, having a common architectural understanding is still work in progress as the high percentage of solution-oriented architectures show that every application context requires their own interpretation. These applications can range from the energy sector [14], health [15], logistics [16] to shop-floor industrial use cases [17] [18] [19]. Most solutions (79) are given by a reference model while 42 define a reference architecture. They differ between these as "a reference architecture extends a reference model by mapping its functionalities onto software elements and the data flows between them". In the following, some selected publications are provided to show the range of approaches and solutions.

When talking about frameworks for the design of DTs multiple scholars (e.g. [18]) and [19]) primarily concentrate on capabilities of a DT rather than proposing concrete solution framework, thus being limited to a requirements viewpoint. The primary capability stated is the ability of retrieving data from physical space, analyzing and processing it in some meaningful way, giving control advice to a human, and apply this control to physical space.

Attempts have been made to create reference architectures of DTs from a functional viewpoint which are left in high levels of abstraction mostly containing building blocks. Work such as Talkhestiani et al. [20] (see Figure 7) and Steindl et al. [21] give a good overview of the building blocks a DT may be composed by, but they fail to be more specific about the interlinks. Thus, they only present an unfinished block definition diagram (bdd) and listing functionalities of DTs (requirements) from their point of view.

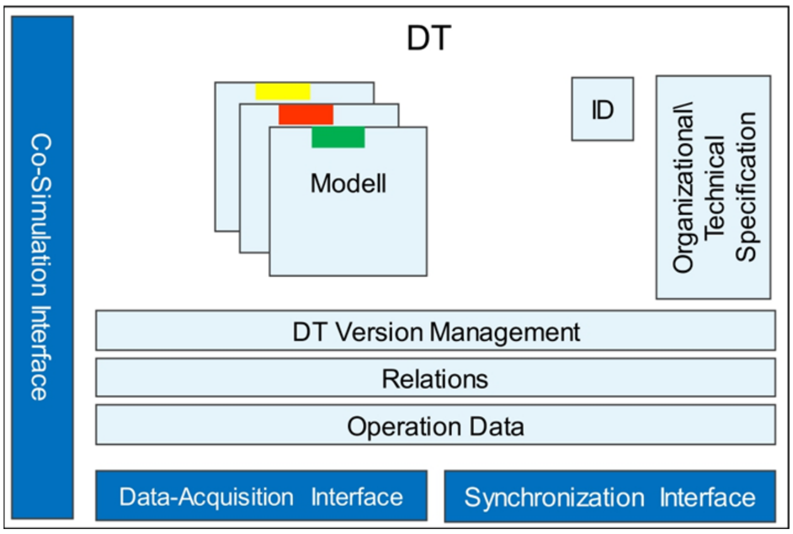
DT
Co-Simulation Interface
Modell
ID
Organizational\ Technical Specification
DT Version Management
Relations
Operation Data
Data-Acquisition Interface
Synchronization Interface

Figure 7 Digital Twin Architecture according to Talkhestiani et al. [20].

Aburru et al. [22] present the most detailed architecture in form of a logical view combined with functional aspects supplying a view of services that are needed in the DT they propose including data flows. This architecture can be seen in Figure 8.

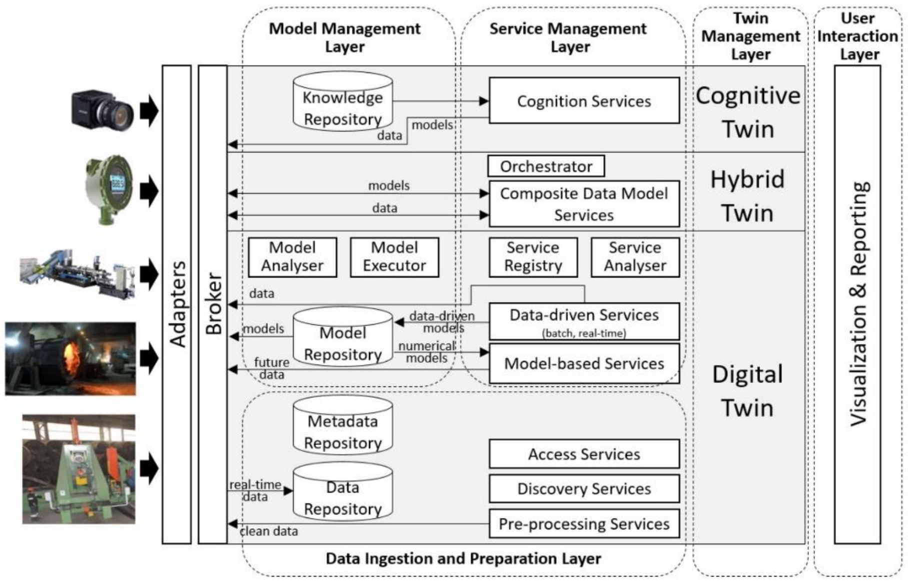

Figure 8 Architecture for a "Cognitive Twin" according to Aburru et al. [22].

A limitation which can be found in all of the above-mentioned architectures is the limitation to describing the architecture from one single VP. Mostly from a requirements/capability perspective or from logical view. However, all-encompassing description combining all stages of the development for a general-purpose DT cannot be found. This makes the applicability of the proposed architecture to other domains difficult since crucial detailed information might be omitted. An exception to this poses Steindl et al. [21] which adopted the IT layer dimensions of RAMI4.0 which can be mapped to VPs as shown in section 2.1.3. However, as already described, the layers/viewpoints in this work are not explained in detail and are described on a high-level. None of the found works apply some common and well-established modeling language hence leaving ambiguity in the interpretation.

In general, the usage of DTs is well established in almost every area related to engineering and a common way to represents these in a generic manner, Therefore, there is an abundance for Digital Twin architectural views which are specific for their application and only a small selection can be showcased in this section. In the following sub-section, a literature review specific on the combination of architectural views for DTs and RL is outlined.

2.3 Literature on DT-based RL

Recent research has delved into the combination of RL and DTs across diverse domains. Osinski 23] introduces an RL training approach through simulations. The domain of application in this case is driving functions for automated vehicles. Nevertheless, it lacks an architectural view and continuous RLA refining. [8] emphasizes training RLA for fusion reactor control with architectural aspects taking a backseat in training consideration. Alexopoulos [24] adopts a DT strategy for robot arm control including architectural specifications, akin to our proposal, but overlooks the post-initial learning. Ju [25] and Gan [26] provide insights into real-world transfer of RL policies and DT-based RL scheduling strategy training, respectively. However, architectural and, yet again, post-deployment learning aspects are under-explored. Similarly, Shen [27] advocates DT and RL fusion for unmanned aerial vehicle control and proposes a DT-based "continuous evolution" in this context [28]. However, the work is domain-specific and generalizable architecture considerations are not present in this work.

In the field of "life-long" RL [29], scholars deal with continuous learning and policy transfer (e.g., from simulation to real world or adjustments from one system to the next). To the authors' best knowledge, availability of training data is assumed in most of the relevant studies. In context of continuous learning, Federated Learning (FL) [30] is a common practice for pre-training and consequent fine-tuning of AI. [30] showcases the combination of DT and FL but leaves the design of these DTs open for interpretation. In general, application independent DT architectures are rare as a high percentage of solution-oriented architectures [31] show that every application context requires their own interpretation.

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

3 Architectural Views of Digital Twins for AI training

There are many aspects to consider when describing the architecture of digital twins. It is therefore important that we distinguish the various perspectives we take when analyzing and describing the reference architecture. Architects need to exercise viewpoint hopping to understand the impact of choices from one perspective to another. In Figure 9 an example is given to clarify this viewpoint hopping: we may step from world to world, and viewpoint to viewpoint. Models from the middle box, the *Data and models* world, determine what happens in the *Simulated world*, where we want to mimic the reality of the CPS. At the same time this simulation has impact in the *Real world* as it is being executed on servers with certain performance and constraints. Changing the type of models has an impact in both worlds.

This type of perspective change can also be seen between the four selected main viewpoints in the remainder of this chapter: requirements, functional, logical, and technical. We have adopted these views from the viewpoints in SPES [12] (see section 2.1.3 for further details). Despite minimizing the number of viewpoints, they can cover all layers and aspects relevant to our approach.

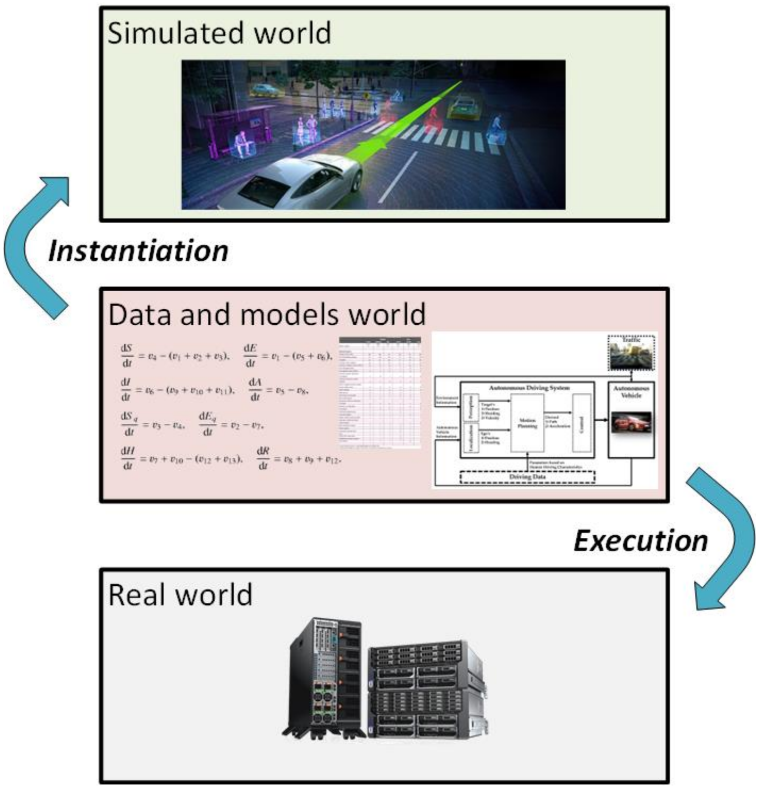

Figure 9 The digital twin can be described using various 'worlds'.

3.1 Requirements View

3.1.1 High-level requirements

The main goal of the ASIMOV approach is finding a self-optimizing solution for a complex CPS. A second goal is the successful application of that solution to similar systems (comparable products or members of a product family), without extensive further development (e.g., re-training the AI from scratch). These two goals are achieved by the cooperation of two main sub-systems: the DT and the AI sub-systems. The DT requirements are derived by looking at the needed behavior and properties in the appropriate phases of use. The DT is mainly operational in the training phase, therefore the requirements on the DT stem from this type of use. We do not consider here the option of using the DT for testing out new settings before applying them to the actual system [5].

We can derive and decompose the requirements from the list shown in the initial ASIMOV reference architecture [5]. The main system qualities mentioned are focusing on the operational phase, so in the case that the AI components have been trained. Here we must reason back from the qualities mentioned at DT-AI-system level in the operational phase, to the requirements for the DT sub-system in the training phase. This influence of the DT on the final set of qualities is rather *indirect*.

- Accuracy of result (i.e., the state of the system after applying the optimization)
  To which extent does the accuracy of the DT influence the accuracy of the total system, mainly the trained AI sub-system? This is a difficult question to answer, as the influence on accuracy is rather indirect. The fact that AI models perform more poorly when using low

accuracy input is not new, e.g., in computer vision as studied by Wang et al. [32]. The fidelity of the DT models does influence the accuracy, but it is hard to determine to which extent since it is linked to the purpose which is very use case, or application dependent. In addition, machine learning algorithms are capable of performing *domain transfer* where one domain would be the task of optimizing a less accurate model of the system and the other being the actual CPS (see [33]). A common example is simulation-trained autonomous driving functions being capable in the real-world of Zhao et al. [34]. If the AI model can bridge the gap, the accuracy of the DT is sufficient. Therefore, the required fidelity must be established in practice, e.g., using experiments, in an iterative development process including the AI sub-system which embeds its validation.

- Robustness

We understand robustness as the ability to manage unforeseen states of the system. Currently we attribute this quality entirely to the behavior of the AI sub-system.

- Reliability

This is the ability to always obtain a good result. Currently we attribute this quality entirely to the behavior of the AI sub-system.

- Reproducibility

We define this as the ability to always obtain the same result. Currently we attribute this quality entirely to the behavior of the AI sub-system.

- Time to result

The calculation time needed by the AI sub-system in the operational phase is independent from the DT. In the training phase, the DT calculation time is obviously very important for the total training time. The faster the better.

- Scalability

This can be defined as the ability to be usable in a product family, and for a variety of usage scenarios. Currently we attribute this quality entirely to the behavior of the AI sub-system.

- Explainability

In short, this is the ability to give insight into how the solution works, and into the plausibility of the result. Currently we attribute this quality entirely to the behavior of the AI sub-system.

The ASIMOV solution has also other practical, realization-related challenges, i.e., non-functional requirements such as [5]:

- Footprint

Extensive DT models and computations require significant resources. These determine also the development costs.

- Integrability

The interaction of the DT with the rest of the CPS has many dimensions: needed resources with possible performance impact, interface definitions, deployment approach (depending on the location of training), etc. Another aspect comes into play when DT sub-components are supplied by different companies. In that case IP protection might be needed, therefore the architecture should be capable of integrating components as black boxes to maintain IP protection.

- Maintainability

Understandability of the DT models is crucial for maintaining and upgrading products in the field. The architecture should be able to plug-and-play new versions of DT sub-components by standardized, e.g., by broadly available connectors. One can also think of this as *containerization*.

- Development cost

This is important for business reasons. The cost can be decomposed into its parts:

  - capital investments (expensive computation tools)
  - operational costs (during development)
  - development effort (time to develop the DT)

Obviously, complex or accurate DT models require much more effort. Therefore, one should take care not to overdo it. Also, extensions of the DT model, e.g., an extra feature or parameter, should scale linearly (at most). This prevents that with increasing complexity, the models do not require unreasonably amounts of effort (quadratic or even exponentially). This is linked to the problem to be solved.

In the training phase the AI sub-system requires data that has certain properties. This is discussed in [35]; it considers the properties of the data generated by the DT, and states that several iterations are needed to understand the data requirements for a given problem. Two main requirements must be considered:

- Representativeness, which indicates to what extent the data represents the problem to be solved. This may involve the diversity of parameters, frequencies, or diversity of errors. This must be determined by domain experts. In [36] an attempt is made to systematically determine the DT parameters that are relevant for data generation. This is, however, a theoretical exercise, yet to be tested on real cases. Some data aspects that fall in the category of representativeness must be mentioned separately, because they are important for achieving good results:
  - Bias, over- or underrepresentation of features in the data.
  - Completeness, getting all aspects and features represented in the data.

- Volume, the amount of data and the time it takes to generate. This challenge, along with the AI algorithms, determines the required amount of data and the training time.

3.1.2 Functional requirements

The functional requirements for the DT were expressed already in an early ASIMOV deliverable [37] using textual descriptions for the two use cases (reproduced here for completeness):

In the TEM case, the DT must:

- model all components and sub-components that contribute to the Ronchigram image formation and the condensor astigmatism.
- have tunable input parameters which are like the tunable settings of the physical electron microscope. In this way, the digital twin can mimic microscope operation when for instance correcting astigmatism. (e.g., source energy, source spot size, aperture diameter, lens current, sample thickness)
- simulate the electron-sample interaction
- model the components that are relevant to the condenser system aberrations and target TEM mode

In the UUV case, the digital twin must:

- have a driving function, represented as if it was in the real system
- reflect the vehicle dynamics
- have LiDAR, radar and camera sensor models
- provide a digital campus environment that the vehicle can drive in

To create a set of generic functional requirements we abstract from the particulars. Therefore, we express the generic functional requirements as follows; the DT must:

- model the components that contribute to the observable output / behavior.
- model the imperfections of the components.

- model the influences of the environment of the physical twin.
- have input parameters that influence its behavior, like the physical twin (so the DT can be exchanged with the PT).

So far, we considered the DT to be a *digital model*, however if we want to extend the DT behavior to a *digital shadow*, or even a *digital twin* [3], we need to add the functionality to synchronize states. Therefore, we add the list of functional requirements:

- synchronize with the PT (the DT follows the state of the physical system).
- optionally, enable the PT to synchronize with the DT (the PT is following in that case).

From this list it is evident that the designers really want to mimic the actual behavior of the physical twin as it is constructed (using the components), and not some idealized system in a perfect environment. This is essential as the ASIMOV solution should work also in the worst cases that occur in industrial reality.

3.1.3 Unobservable parameters

It might be beneficial to exploit an additional value of a DT: export the ‘unobservable’ parameters! These parameters or data are not available in the physical twin (a real CPS), but the simulated model has access to all variables. The AI may use them to its advantage, because effects can be observed in the data effects earlier in time, or more clearly.
In various modelling paradigms, the unobservable parameters or counterparts can be found. To start with, note that reinforcement learning is often stated in the context of a Markov Decision Process, where the *Markov* property means that the most recent (observed) state is informative enough so that it contains the information from the full history; *Decision* is where an agent can take action, and the *process* is about having discrete time steps.

In Markov Decision Processes, the state can be *partially observable* or *inexact.* Partially observable means that the state is e.g., given by the temperature distribution of the TEM, but only temperature on one position (e.g., somewhere inside the room the TEM stands in) is observed; inexact means that there is e.g., considerable measurement noise. Control theory deals with estimating the true, unknown state of a system whereas in a DT this true state is available. Another example where decisions are taken daily is in greenhouses, based on weather forecasts for the next day. During development, the forecasts can be replaced by realizations, so that forecast errors are temporarily eliminated. Unobservable variables are quite common among statistical models which may be useful if a statistical model is used in the DT.

3.1.4 Variation and variability requirements

The second goal mentioned in 3.1.1, i.e., robustness, requires the DT to change its behavior according to the expected distribution in a product family, or when synchronizing with the PT. Another type of variations in systems may be expressed as a continuous change (properties change slowly over time) or a discrete change (abrupt change in system settings, e.g., by a system mode switch). Both changes can be modeled using the configuration parameters, in section 3.3.1.2 indicated by ‘*c*’.
In the reference architecture, there is a dedicated component assigned to inserting variation, see sections 3.3.3.3 and 4.4.3.

3.2 Functional view

The functional view is probably the most fundamental view for understanding what a system does. It should describe and explain the transformation of the inputs into the outputs. We start - for simplicity - at the very top of the functional decomposition tree: the main function of the product. Next, we decompose the main function into sub-functions, as we want to show the functions of the Digital Twin.

We use the easy-to-understand IDEF0 standard [38] to express the functional diagrams.

3.2.1 Functional model for the TEM case

A high-level view on the main function of a TEM is shown in Figure 10. The function F1 is expressed as a verb-phrase capturing the global function of the system.

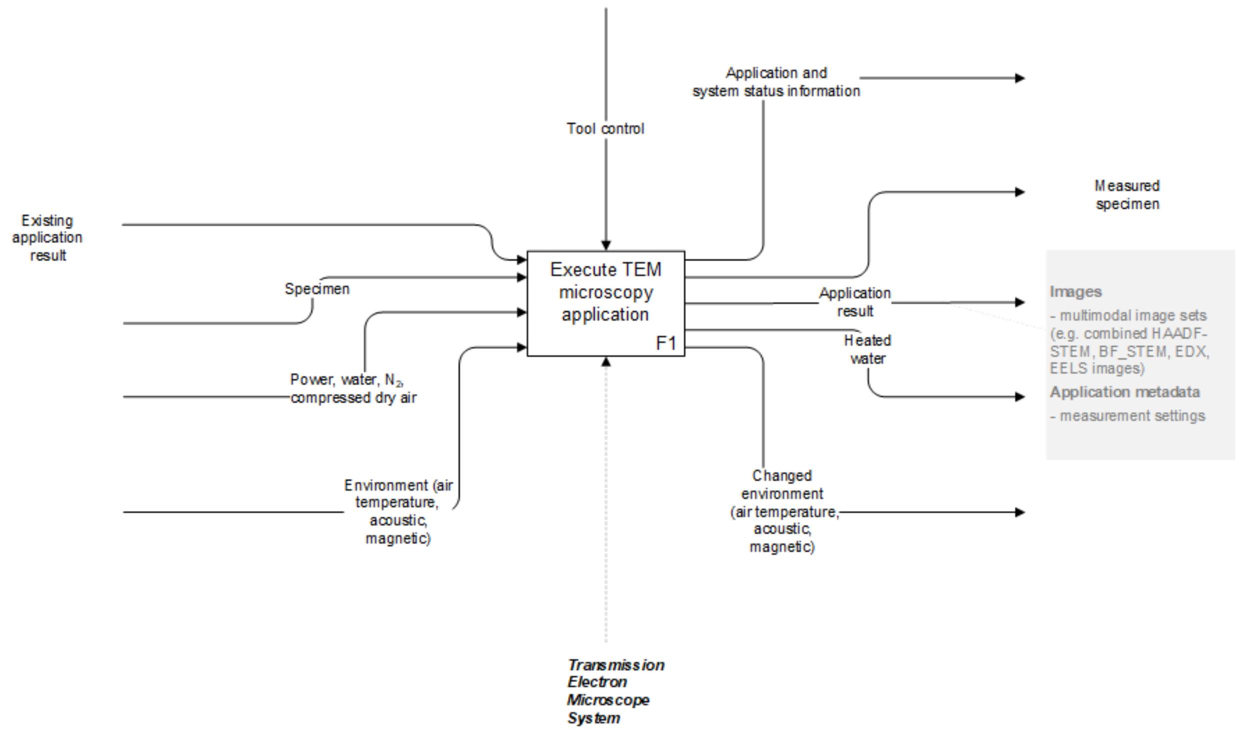
Tool control
Application and system status information
Existing application result
Specimen
Power, water, N_2, compressed dry air
Environment (air temperature, acoustic, magnetic)
Execute TEM microscopy application
F1
Measured specimen
Application result
Heated water
Changed environment (air temperature, acoustic, magnetic)
Images
- multimodal image sets (e.g. combined HAADF-STEM, BF_STEM, EDX, EELS images)
Application metadata
- measurement settings
Transmission Electron Microscope System

Figure 10 Functional model of a TEM.

A functional diagram of the larger context of a digital twin and AI combination for the TEM case is shown in Figure 11. In the operational phase, an outside agency (e.g., a microscope operator) indicates the need for an optimization of a certain system setting. This information is applied as an ‘optimization goal’, shown at the top of the figure. This will start the optimization procedure in F3, by giving out the appropriate commands and settings to the DT and the AI functions. When the optimization goal is reached to a sufficient level (as specified in the optimization settings), F2 will stop the optimization procedure and signal its completion.

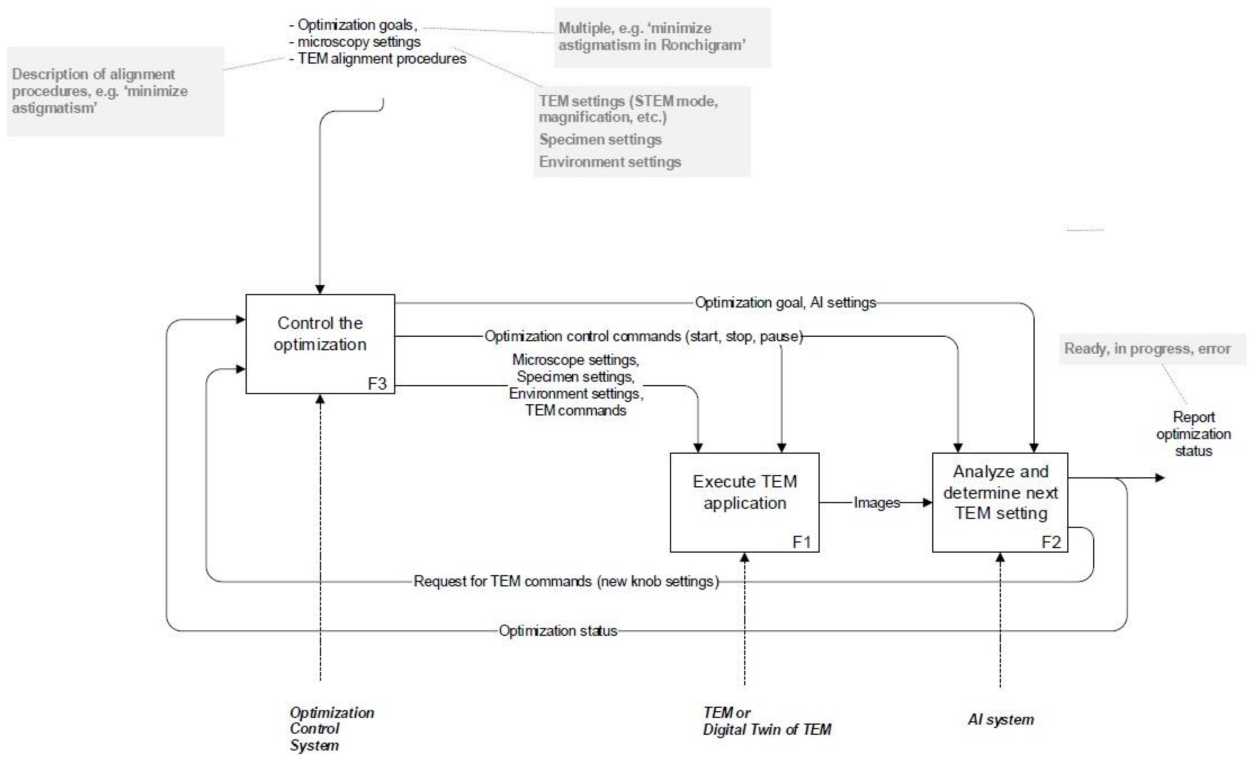
- Optimization goals,
- microscopy settings
- TEM alignment procedures
Multiple, e.g. 'minimize astigmatism in Ronchigram'
Description of alignment procedures, e.g. 'minimize astigmatism'
TEM settings (STEM mode, magnification, etc.)
Specimen settings
Environment settings
Control the optimization
F3
Optimization goal, AI settings
Optimization control commands (start, stop, pause)
Microscope settings,
Specimen settings,
Environment settings,
TEM commands
Ready, in progress, error
Report optimization status
Execute TEM application
F1
Images
Analyze and determine next TEM setting
F2
Request for TEM commands (new knob settings)
Optimization status
Optimization Control System
TEM or Digital Twin of TEM
AI system

Figure 11 Functional diagram for the operational mode in the TEM case.

3.2.2 Functional model for the UUV case

The functional diagram for the UUV case is shown in Figure 12. Nearly all information flows are analogous and can be translated, except for the Specimen settings. The environment plays a much more important role in the UUV case. One might say that the specimen is part of the 'environment' in the TEM case.

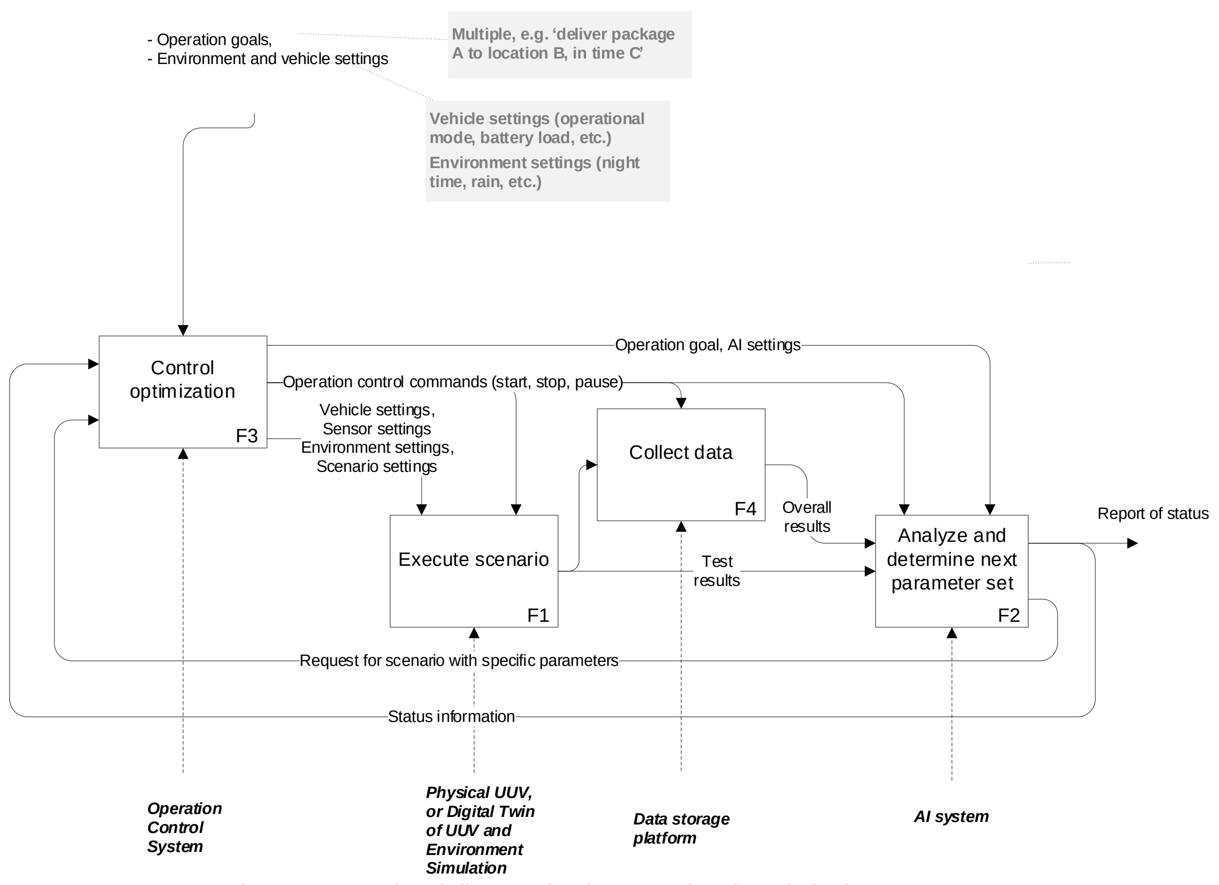
- Operation goals,
- Environment and vehicle settings
Multiple, e.g. 'deliver package A to location B, in time C'
Vehicle settings (operational mode, battery load, etc.)
Environment settings (night time, rain, etc.)
Control optimization
F3
Operation goal, AI settings
Operation control commands (start, stop, pause)
Vehicle settings, Sensor settings Environment settings, Scenario settings
Collect data
F4
Overall results
Execute scenario
F1
Test results
Analyze and determine next parameter set
F2
Report of status
Request for scenario with specific parameters
Status information
Operation Control System
Physical UUV, or Digital Twin of UUV and Environment Simulation
Data storage platform
AI system

Figure 12 Functional diagram for the operational mode in the UUV case.

The difference can be found in the collection of data function (F4). This function is needed to deliver the *historic data* for the analysis in function F2.

In the Execute Scenario block, the co-simulation platform Model.CONNECT is triggered via Flask to execute the simulation, which integrates many related components, in ASIMOV including mainly driving functions, driving dynamics and the Carla interface [39]. When executing the simulation, Carla will be launched, which will not only create a realistic 3D environment, but also implement some sensor models on the ego car. The simulation results will be analyzed in post-processing and then used for Reinforcement Learning. Below are descriptions of the mainly used modules of Model.CONNECT and Carla:

- Driving Function provides basic Adaptive Cruise Control (ACC), as well as Autonomous Emergency Braking (AEB) and Lane Keep Assist (LKA), which provide enough functionality to test the desired scenario variation effects.
- Driving Dynamics is provided by VSM, a comprehensive, flexible and real-time simulation tool. It employs a non-linear multi-body approach that accurately reproduces physical driving behavior, including longitudinal, lateral and vertical dynamic effects.
- Carla Interface relies on Python script, which can create 3D environment and implement sensor models to capture the simulation environment Information around the ego vehicle.
- Sensor models like Camera, Lidar and Radar are integrated in Carla. Camera is used for lane detection, while the other two are responsible for ACC and AEB.

3.2.3 Generalized functional model

We derive the generic functional diagram from Figure 11 and Figure 12 to be part of the reference architecture. Although there are some differences, e.g., not all inputs and outputs are similar, we still think the overall structure is the same, see Figure 13.

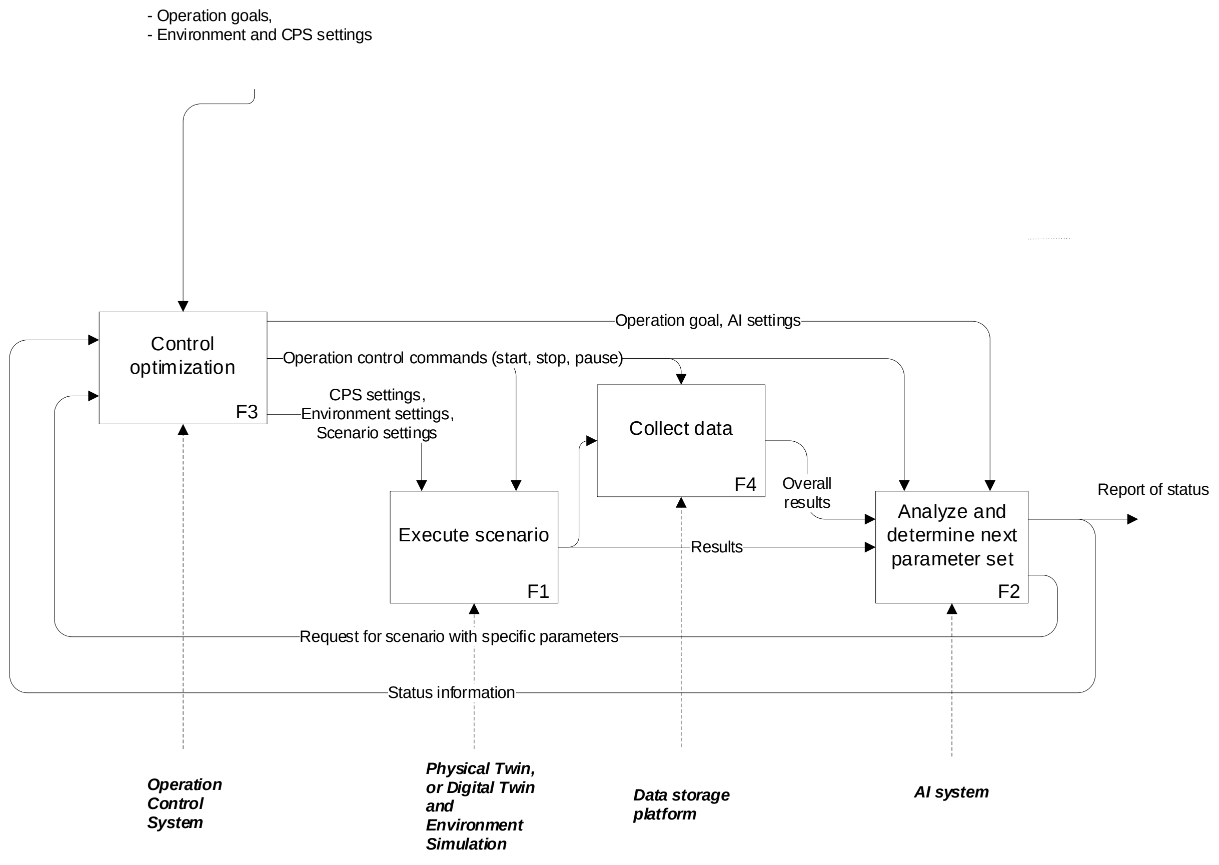
- Operation goals,
- Environment and CPS settings
Control optimization
F3
Operation goal, AI settings
Operation control commands (start, stop, pause)
CPS settings, Environment settings, Scenario settings
Collect data
F4
Overall results
Analyze and determine next parameter set
F2
Report of status
Execute scenario
F1
Results
Request for scenario with specific parameters
Status information
Operation Control System
Physical Twin, or Digital Twin and Environment Simulation
Data storage platform
AI system

Figure 13 Generic functional model of a CPS with a DT+AI optimization functionality.

3.2.4 Generalized functional model of the DT

Decomposing the generic model of function F1 requires a split of the control functionality and the 'payload' functionality. The latter is the actual functionality to be calculated or simulated. Figure 14 shows a typical example in which the functionality could be split into two features that can be simulated separately in sequence. Note that feature 2 uses the results of feature 1.

At this point we see the effect of modeling physical reality. Is it possible to use separate models that describe the physical processes, for instance along a sequence or pipe-line architecture, or as a fully parallel set of processes? The functionality that needs to be simulated by the DT determines entirely how the functional structure looks like. E.g., in the TEM case the flow of electrons determines the DT functional structure (see [37]).

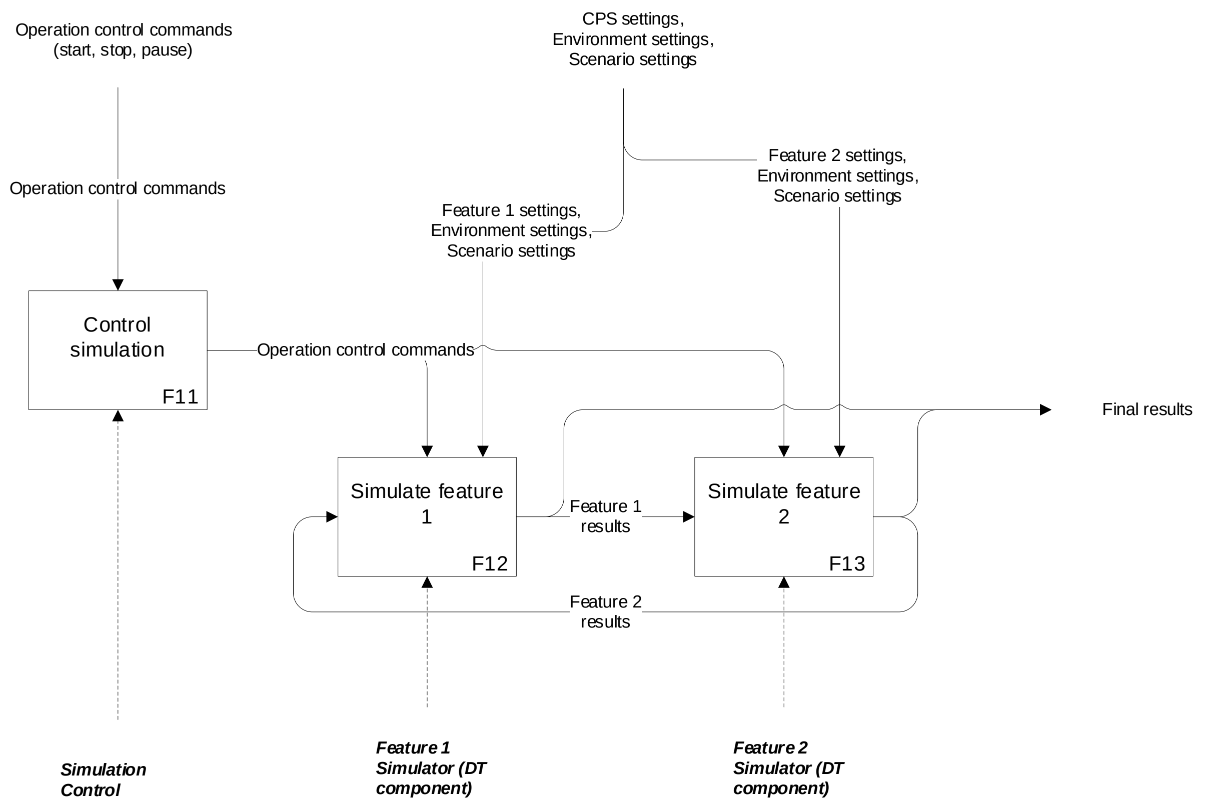
Operation control commands (start, stop, pause)
Operation control commands
CPS settings, Environment settings, Scenario settings
Feature 2 settings, Environment settings, Scenario settings
Feature 1 settings, Environment settings, Scenario settings
Control simulation
F11
Operation control commands
Final results
Simulate feature 1
F12
Feature 1 results
Simulate feature 2
F13
Feature 2 results
Simulation Control
Feature 1 Simulator (DT component)
Feature 2 Simulator (DT component)

Figure 14 Example of the generic functional model of the DT for two separate features. Note that the second function, F13, uses output of F12, and vice versa.

Note that we left out (for simplicity) the 'diagnostic information' output, as mentioned in the WG reference architecture [5].

3.2.5 Workflows and states

The following activities for the DT can be foreseen:

- DT being used for training the AI
- DT being used for checking the impact of a parameter change before applying to the physical twin
- DT being used for synchronizing with the physical twin
- Updating the DT models

These purposes can be executed using only three states, as shown in the state diagram in Figure 15. The behavior in these states is straightforward:

- In the *Idle state* the DT waits for instructions
- In the *Calculating state* the DT is calculating/simulating (upon input of settings and providing output after finishing the calculations)
- In the *Updating state* the DT receives new calculation models, or settings (e.g. twinning).

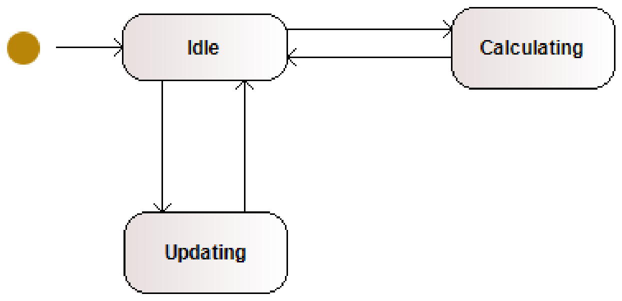

Figure 15 State diagram for the DT.

3.3 Logical View

As described by [12], the logical viewpoint supports the design of a solution to the functional view in terms of communicating components in a component architecture. In this section, the component architecture developed over the project will be explained first as a basis for the following description of the components and their composition.

3.3.1 Current work in ASIMOV

Until now, during the project, multiple views containing componentized architectures have been developed. Primarily, in the work of the ASIMOV workgroup, as well as in T2.2: “Methods and Tools for Training AI with Digital Twin”. In the following three main concepts that need to be integrated in the architecture are mentioned.

3.3.1.1 Inside of DT

Figure 16 shows the Reference Architecture developed in the ASIMOV Working Group. To examine what the inside of a DT could look like, we first have to look at the DT in context with the remaining architecture components.

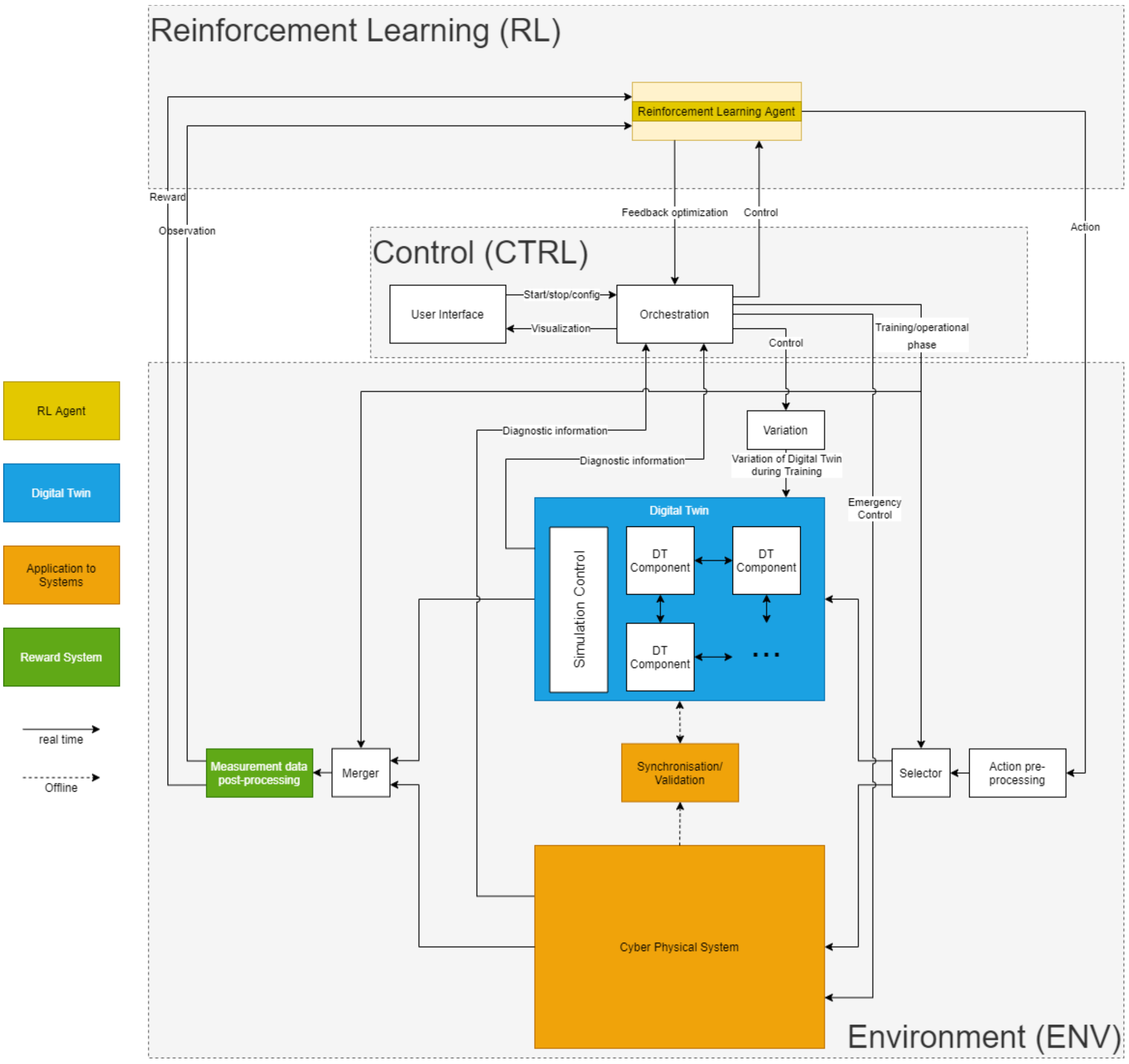

Figure 16 ASIMOV Reference Architecture developed by the Working Group.

The main interfaces to DT surroundings are control inputs, which are connected to some orchestration, an interface for introducing variations, which are further described in Deliverable D2.2.1, as well as pre- and post-processing of data from/to the interface with the Reinforcement Learning Agent (RLA). The final interface is a connection to the Cyber Physical System i.e., the Physical Twin (PT) via a Synchronization/Validation block.

The DT itself can be subdivided into multiple components. First, depending on the complexity of the system, it is very likely that multiple simulation tools or models are used to replicate accurate behavior of the whole system, each detailing another aspect of the system. We call these DT Components. There are multiple ways that these subcomponents can work together. A hierarchical connection between DT components can be beneficial for one use case, while a non-hierarchical connection might be better for other use cases. The division into different DT Components can follow a pattern that resembles the components of the physical system, should the alignment and explainability of the individual components behavior be of interest. It can also follow a more performance-oriented division, based on the used simulation tools, trading analogy to the real system for computational efficiency.

Independent of the DT Component structure, the combination of multiple DT Components requires a simulation control for managing the interaction between the components. This can be represented by a

Co-Simulation-Platform that manages interfacing between several simulation tools, or simply the connection of multiple functionally encapsulated Sub-Models inside an overall model in program code.

The division of the entire DT into DT Components is essentially highly dependent on the use case, but there are some benefits to a well-chosen structure. This is especially important, when the connection and comparison to a real system comes into play. If the DT Components resemble actual subcomponents in the PT, component-wise data comparison is possible. This enables tracking down misalignments and fine-tuning opportunities to component level, which can bring component expert knowledge into play and lead to better explainability. This is further elaborated in Deliverable D4.1 [40].

3.3.1.2 Adaptation of DT

The DT has, in contrast to a conventional model, the ability to adapt its behavior over time to stay aligned to its Physical Twin (PT) counterpart. We call this specific type of DT operation *Digital Shadow*. The ability to adapt its behavior can be achieved by a connection to the PT in combination with an additional module that compares DT and PT behavior and fine-tunes the behavior of the DT to match that of the PT. The following section will explain Figure 17 and give an overview of how such an adaptive DT architecture can be achieved.

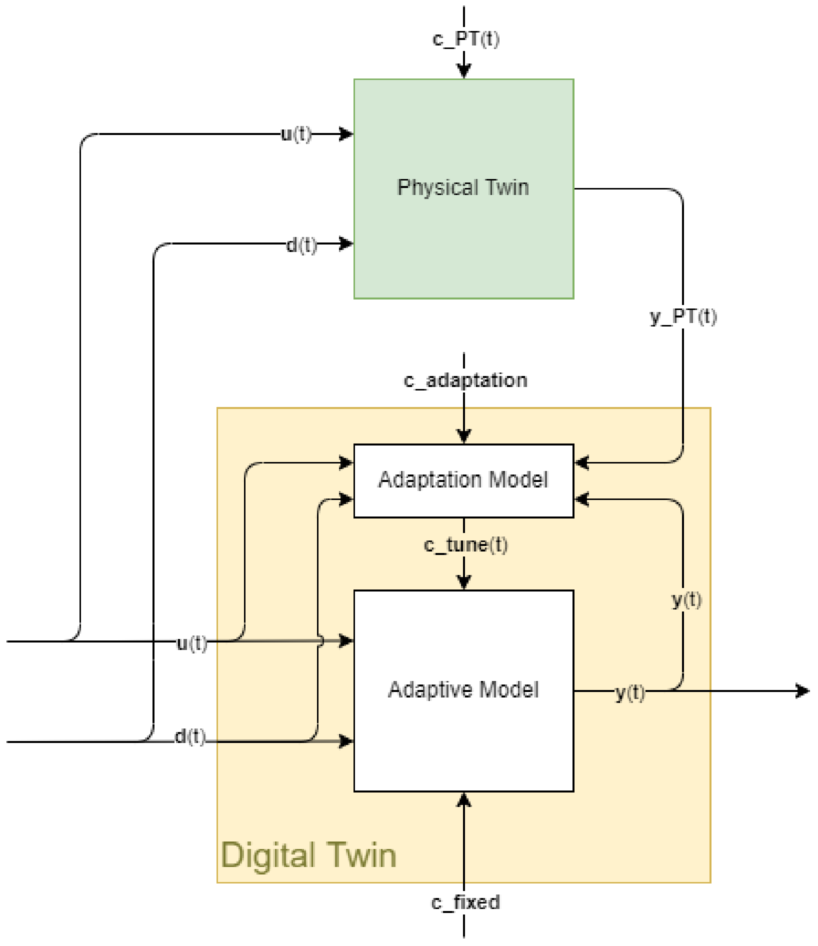

Figure 17 Adaptive Digital Twin component overview.

The DT as well as the PT both receive $u(t)$ and $d(t)$ as inputs. In this case, $u(t)$ are all the controllable inputs into a system and $d(t)$ are all additional inputs that are not controllable, but still influence the system. These will be called disturbances in the following sections, even if the exact definition of disturbances might differ from that. The output of both systems is $y(t)$ and $y_{PT(t)}$ respectively.

The PT can be seen as a system that receives a defined number of inputs, combined with a possibly infinitely large number of disturbances. These disturbances contain every environmental input into the system, which influences the output of the system. The configuration of the PT $c_{PT(t)}$ can be seen as a potentially infinitely large set of parameters that define the correlation between input and output. That's where the "logic" of the system is located. This logic can be expressed in differential equations combined with parameter values. In general, all the inputs, as well as the configuration of the PT are time dependent.

As the PT is dependent on a potentially infinitely large set of parameters and inputs, it is not possible to create an identical copy of it as a DT. Only a subset of all disturbances, I.e., the ones that appear in the chosen abstraction of the system during model creation and have been measured, can enter the DT.

The DT comprises two sub-modules, the Adaptive Model and the Adaptation Model. The Adaptive Model can be seen as the part of the DT, which is quite like a conventional model. It receives $u(t)$, as well as the subset of $d(t)$ that it is receptive for, and a fixed parameter set c_{fixed} to calculate its output $y(t)$. Note, that c_{fixed} is not time-dependent, but initially configured during setup of the adaptive model. The model's behavior therefore does not change over time but is only dependent on $u(t)$ and $d(t)$ once the simulation is running.

The Adaptation Model is what transforms the conventional Model into a DT. It can be seen as a module, which analyzes the inputs and outputs of the PT and the DT and counteracts misaligned behavior through parameter fine-tuning in the DT. The parameters that can change for that realignment are $c_{tune(t)}$. This is a subset of all configuration parameters of the Adaptive Model, which have been chosen to be adaptive during conception of the DT. Parts of the Adaptive Model that shall not change, can still be defined in c_{fixed}.

The behavior of the Adaptation Model and therefore its logic on how misalignments between DT and PT shall be fixed, as well as certain thresholds, can be configured via the $c_{adaptation}$ parameters, which are also time-independent.

An example of such an Adaptation Model could be a wear model, which describes how mechanical parts deteriorate over time. It analyzes the behavior of the PT to find a fitting set of parameters to use in the wear model which influences the DT.

It must be noted, that as the Adaptation Model is configured before running the DT, which limits the scope of how the DT can adapt to what is measured. Unexpected effects could still be covered by a data-driven model inside the adaptation model at the expense of explainability.

The interface of the Adaptation Model to the Adaptive Model can also be used in the process of creating variations as introduced in T2.2. Variations can be seen as instances of DTs that are not connected to a specific PT, but have their parameters set instantiated, based on parameter distributions. That way a virtual fleet of diverse DTs can be used for training the Reinforcement Learning Agent and improving generalization.

3.3.1.3 System configuration variation

In the deliverable D2.2.1, a special view on the interaction and development of the RLA from the training phase to the operational phase has been established (see Figure 18). The primary idea is the existence of a reference RLA with a Digital Twin prototype which are instantiated in the training phase with variance in configurations to improve generalization of the reference RLA's policy. It also expands to the operational phase where the RLA instances train on a Digital Twin of a Real system in order to provide optimization suggestions and to fine-tune to the given instance of the real system. For a more detailed explanation, consult D2.2.

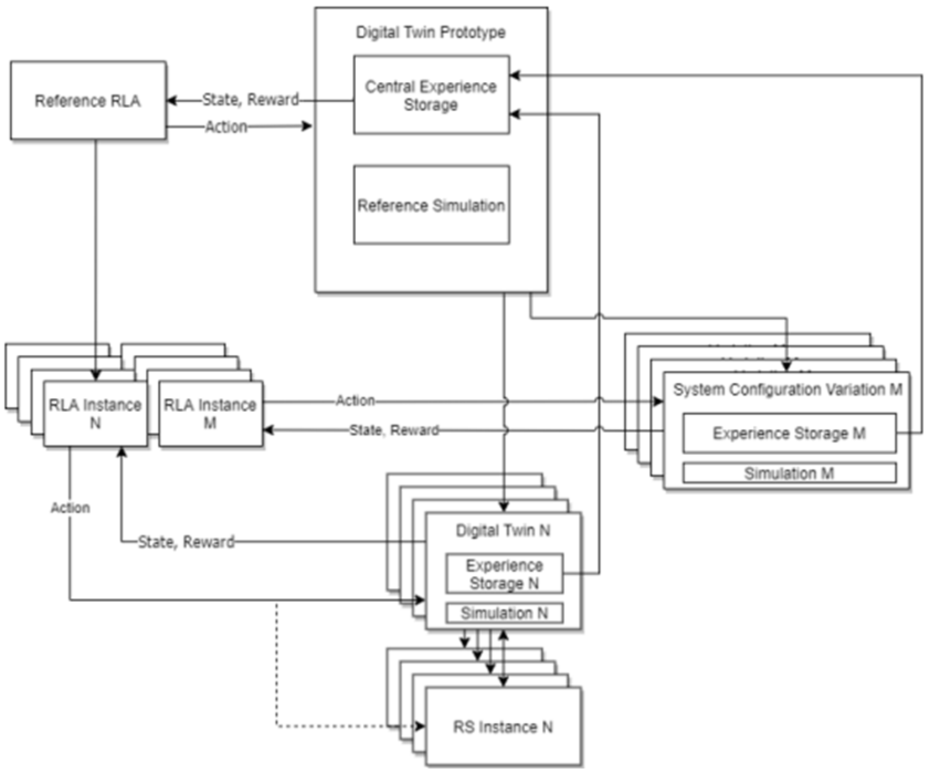

Figure 18 Training process using system configuration variations and Digital Twins.

This view indicates that the Digital Twin needs to be capable of interacting with the RLA in the form of action, state and rewards which is in line with reference architecture from the ASIMOV workgroup. It is indicated that storage and simulation is part of the Digital Twin (Prototype) and a possibility of interaction with the real system (RS) is needed (also found in the reference architecture) but also that the during the DT training phase, the DTs can be turned into System Configuration Variations: a model which represents a fictitious instance of a real system.

3.3.2 ASIMOV phases

The adaptation of the DT and its composition out of multiple small sub systems must be reflected in an overall System Composition. Furthermore, this System Composition shall align with the previous work done and captured by the working group in the reference architecture. To accomplish that, the ideas of the previous sections have been captured in SysML-conform diagrams.

SysML[1] is a language for model-based system development which is based on UML but has been expanded to be fit for purpose. The benefit for using such a language is clear: a standardized language avoids ambiguity in interpretation and is easily readable by many people. SysML is a language specifically designed to develop any system, however, it is especially recommended for DT creation (see [41]).

For now, we distinguish between the training and operational phase. In the training phase, the ASIMOV solution consists of three types of components as it can be seen in Figure 19: a controller, a simulation environment and a reinforcement learning agent. While there is just one controller which has the task of the orchestration of the training and providing an interface for humans, there is no restriction whether there is one or multiple simulation environments present in the ASIMOV solution during that phase. This

[1] https://sysml.org/

is for enabling parallelization of the training with their corresponding instance of a RLA as shown in Configuration Variation Process.

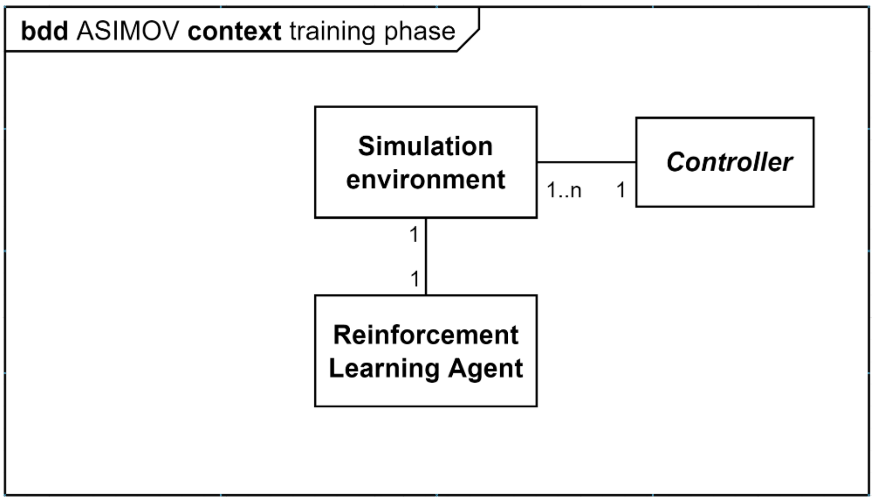
bdd ASIMOV context training phase
Simulation environment
Controller
1..n
1
1
1
Reinforcement Learning Agent

Figure 19 Block definition diagram of the ASIMOV solution during the training phase.

In the operational phase, the arrangement does initially not change, but a cyber-physical system (CPS) or also called Physical Twin is associated to each simulation environment where an instance of its Digital Twin resides (see Figure 20). Also, it is a question of implementation and requirements from the stakeholder whether one CPS requests its optimization, or it is management by a higher-order management system which may control the state of multiple systems. Thus, a relation between a CPS, Simulation environment and RLA tuple is denoted but no numbering.

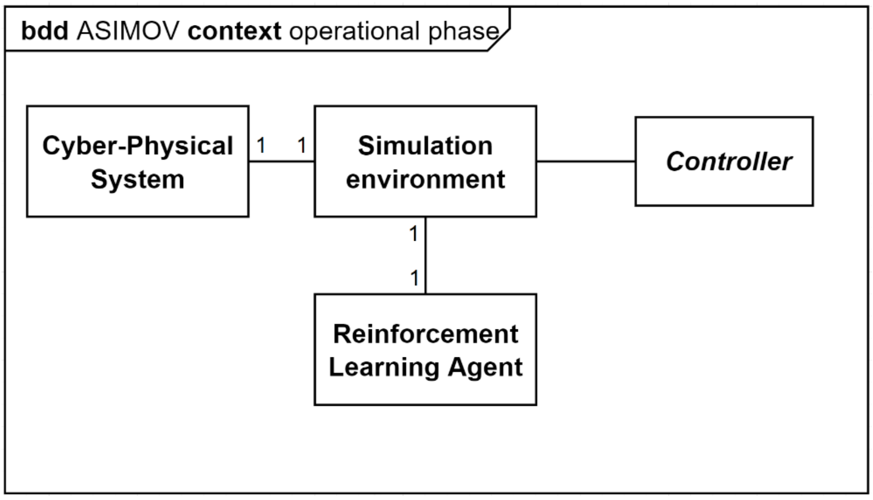
bdd ASIMOV context operational phase
Cyber-Physical System
1
1
Simulation environment
Controller
1
1
Reinforcement Learning Agent

Figure 20 Block definition diagram of the ASIMOV solution during the operational phase.

While it can be specified which components are needed for the ASIMOV solution, the composition and interlinks depend greatly on the type of RL algorithm used. While simpler algorithms like methods of Temporal-Difference Learning update their policy after each step, actor-critic approaches use a 2-step process. In the first step, the actor samples several episodes using a simulation environment (also in parallel) which are stored in a “experience storage” and in a second step the updating of the critic and actor is performed using the sampled episodes. For this reason, an internal block diagram of the ASIMOV solution is not shown in this document as it is also out of this document’s scope.

3.3.3 System composition

3.3.3.1 Simulation Environment

The Simulation Environment is the main container for everything needed to generate data for the RL training. It first consists of a single Control block, which serves as a communication interface with the remaining components of the ASIMOV solution. It must be able to start, stop and reset the Digital Twin, control when and where simulation data is stored and control the Simulation. It could be realized via simulation program or a co-simulation platform. It is subordinated and takes commands from the controller of the training but also provides feedback.

It furthermore comprises the DT itself, being the digital representation of the system of interest. On this level, we can additionally state that a single DT is part of a simulation environment. More complex representations of DTs can be realized via a hierarchical conjunction of multiple DTs. This will be further explained in the next section.

An arbitrary number of databases is also part of the simulation environment. Depending on the type of data and the data necessary to either train the RLA or adapt the DT, multiple different databases can be used here.

As the DT supplies some insightful output in any form, it might not be adequate to supply good training data in the format of a reward and state signal. Thus, post-processing needs to be part of the environment and is already established in the ASIMOV reference architecture. The same holds for the input where the action of the RLA needs to be preprocessed before being able to be used by the Digital Twin as an input.

As already stated, the simulation environment needs to be capable of twinning the cyber-physical system. Thus, a component of this task needs to be part of the environment which can interact with the physical system and retrieve information.

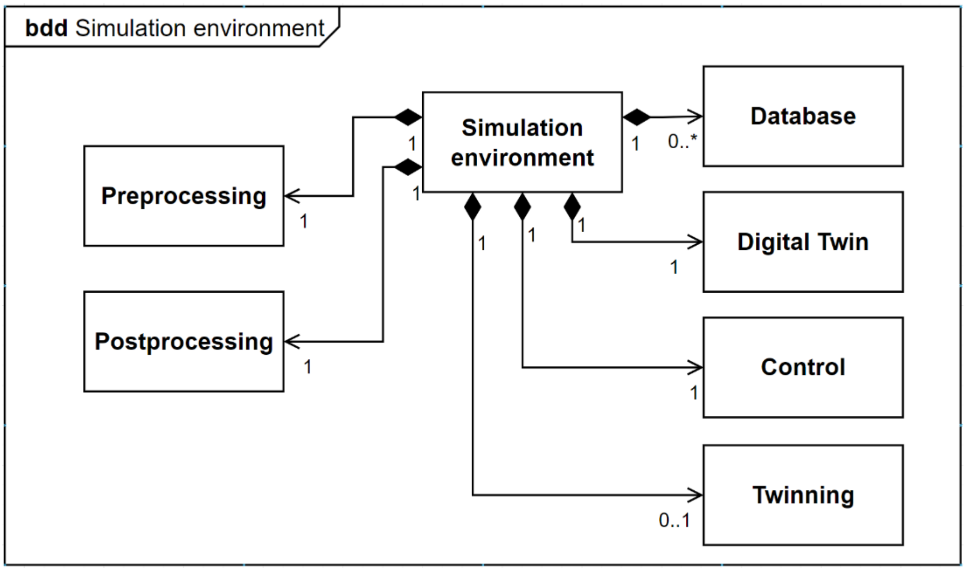

Figure 21 Block definition diagram of the simulation environment.

Based on the reference architecture from the work group and the capabilities and duties of the system outlines in the functional view, an internal block diagram can be created showing the connections between the building blocks described above. The result is shown in Figure 21.

The simulation environment, as illustrated in Figure 22, interacts through five ports: (i) control input from the controller that manages the RL training, (ii) connection to the CPS and (iii) action, (iv) reward and (v) status ports for communication with RLA. The flow of data is from the action port via a pre-processing component into the DT as simulation input and after computing, from the output of the DT via a post-processing component into the state and reward output of the simulation environment. The pre- and post-processing components convert the RLA/DT output into a suitable format. A controller manages the processing and optional databases that provide additional information. In the case of the operational and fine-tuning phase, a twinning block is implemented to orchestrate the retrieval of CPS data according to the requirements. Since no physical counterpart exists during the training phase, it is replaced by a variation generator within the DT that mimics the existence of multiple, slightly different CPS instances.

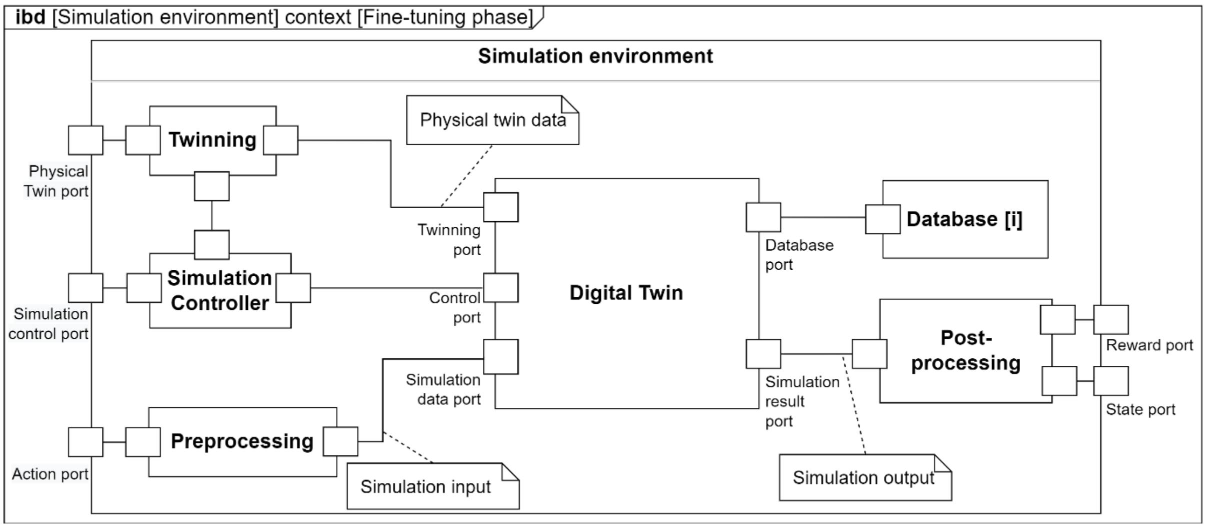

Figure 22 Internal block diagram of components and connections that compose the simulation environment .

Notable is the fact that the simulation environment consists of one DT. We see DTs capable of being hierarchical: one DT can consist of multiple DT analogous of a system which can consist of multiple components which in fact are also systems.

In contrast to the operational phase, in the training phase no CPS exists from which to retrieve output data for twinning, the updating of the DT. Thus, the component “Twinning” and the physical twin port is only active or implemented during the operational phase.

3.3.3.2 Digital Twin

The DT Block, used in Figure 22 can be further split-up as shown in Figure 23.

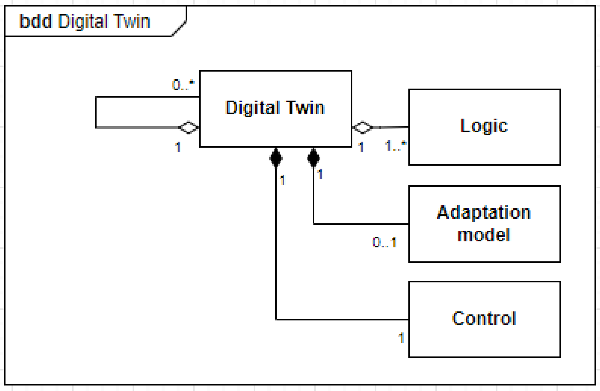

Figure 23 Block definition diagram of a DT.

A Digital Twin is an aggregation of at least one Logic Block and composed by one Control Block. The Logic Block is where calculations happen. An arbitrary number of inputs is translated into an arbitrary number of outputs. The block represents what a conventional model would do.

Like the control block in the simulation environment, the control block inside the Digital Twin manages its underlying components. This mainly involves a correct control of when a component shall be updated in terms of realigning it to the PT and transmitting this information to the respective components of the DT. Other tasks might involve the synchronization and managing start, stop and reset of blocks. The controller is subordinated and takes commands from the controller of the simulation environment but also provides feedback.

The Adaptation Model serves two purposes in this DT. If the Logic(s) of this specific DT shall be adapted to its physical counterpart, it outputs the new configuration parameters, as shown in [Figure 12]. If the DT itself contains other DTs, the adaptation model can provide the correct PT data for the adaptation models of sub-DTs. How this is done is a matter of research.

This enables the possibility to link multiple different DTs together as shown by the self-reference of the DT Block in Figure 23. This possibility is crucial for the usage of DT of system components that might have originated outside, e.g., supplied by the manufacturer of the component. This enables possibilities such as IP protection or decentralized simulation.

The design of an internal block diagram of a DTs is more challenging since multiple logic blocks can be linked to "sub"-DTs and vice-versa. The result can be seen in Figure 24. As already described in the specification of the environment simulation, a DT possesses five ports to interact with other components within the simulation environment: Twinning port, Control port, Simulation data port, Database port and Simulation result port. Via the Twinning port output data of the physical twin (CPS) is passed to the adaptation model whose function is to update the model's configuration accordingly. Its functionality will be explained in more detail in section 3.3.3.3. Its output is two-fold: on one hand, it passes the data from the physical system through to DTs it is responsible for. On the other hand, it computes adequate configuration parameters for the Logic blocks based on the comparison of PT data and simulation results. This adaptation is controlled/triggered by the DT Controller who also interacts with super- and subordinated controllers.

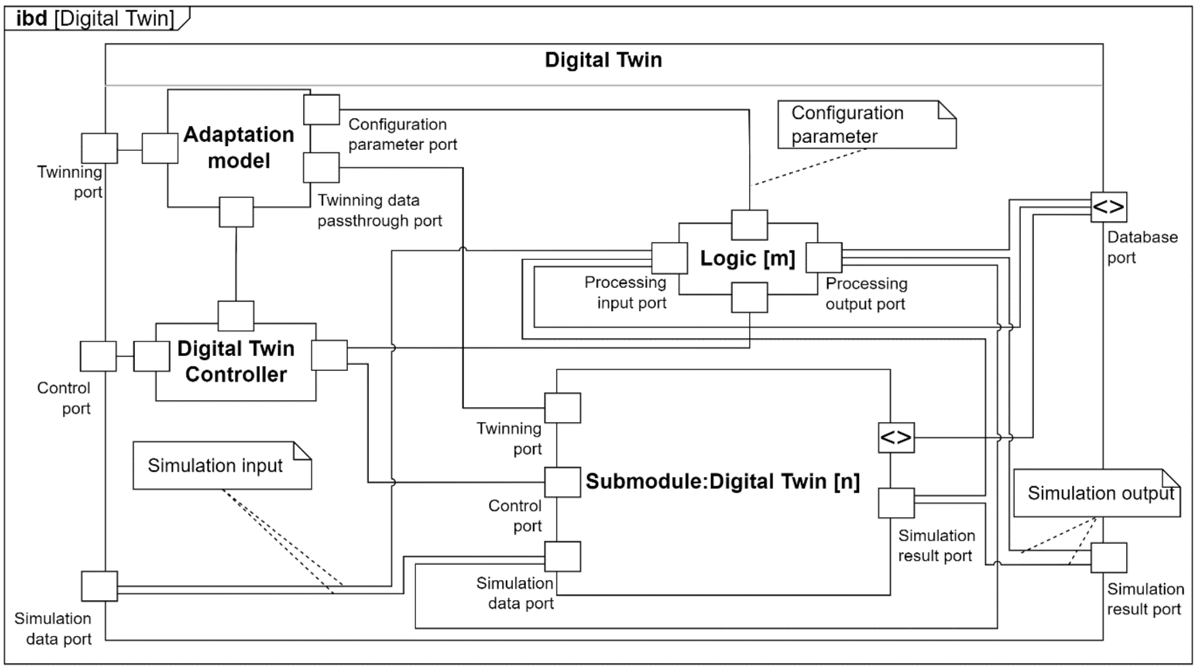

Figure 24 Logical components and communication of the DT

Simulation data from outside the DT is passed to the Logic blocks and DTs, as shown in Figure 24. The logic blocks can send and receive data from databases. Simulation/Computing results from both DTs and Logic blocks can be the input to other Logic and DT blocks, as a feedback form the processing output port and simulation result port to the processing input port and simulation data port of the Logic blocks and DTs, respectively. The logic blocks can calculate the behavioral output of the DT by using simulation data input to the DT itself, simulation data output of hierarchically nested sub-DTs, static configuration parameters sourced from persistent databases, and dynamic configuration parameters obtained through the adaptation model.

3.3.3.3 Figure 24 Adaptation model

The adaptation model's purpose is to set up the "model" (composed by Logic(s) and DT(s)) in such a way that it mimics the behavior of a/the physical counterpart good enough for training, fine-tuning or verification of the optimization suggestion. The adaptation model is the component whose nature changes the most depending on the phase of the ASIMOV solution is in. During the training phase it possesses the capability to generate a "imaginary" variant of the system (sometimes referred to as "System Configuration Variation") by randomizing configuration parameters or setting them based on some rule (e.g., for improving generalization/coverage of sample data). This is done without having an actual physical system as a counterpart, thus without a connection or data from one. During the operational and fine-tuning phase, the DT shall represent an actual system. To provide this capability it takes the output of the physical twin given a reference input and calculates the necessary configuration parameters to fit this behavior. Since a DT can be composed of other DTs it shall pass the relevant information on physical twin data to its subsidiary DTs. These requirements in capability give place to the following block definition diagram of the Adaptation Model as seen in Figure 25.

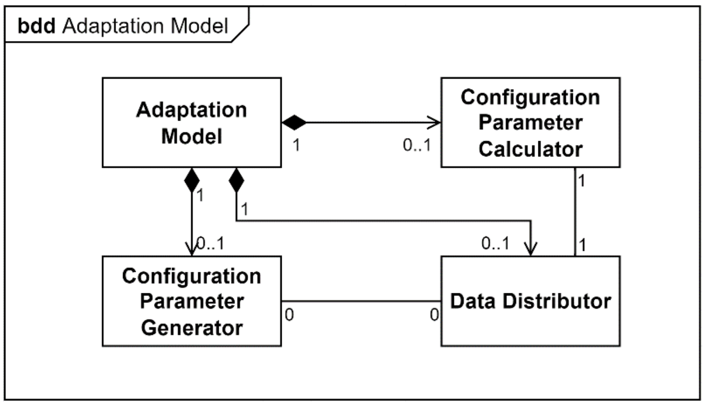

Figure 25 Block definition diagram of the Adaption model showing its composition.

An adaptation model can be composed by one "configuration parameter generator" during the training-phase or Configuration Parameter Calculator and Data Distributor for the operational and fine-tuning phase. If either of the latter exists, then the other is present as well.

With respect to the internal connection of blocks, the adaptation model must be capable of receiving the relevant data from the Physical Twin y_{PT}). It also obtains control inputs which trigger the updating of configuration parameter and feedback is supplied. As output the adaptation model, of course, needs to be able to pass configuration parameter to the logics and twinning data to subsidiary digital twins. Combined with the blocks as defined above, an internal block diagram as in Figure 25 can be designed.

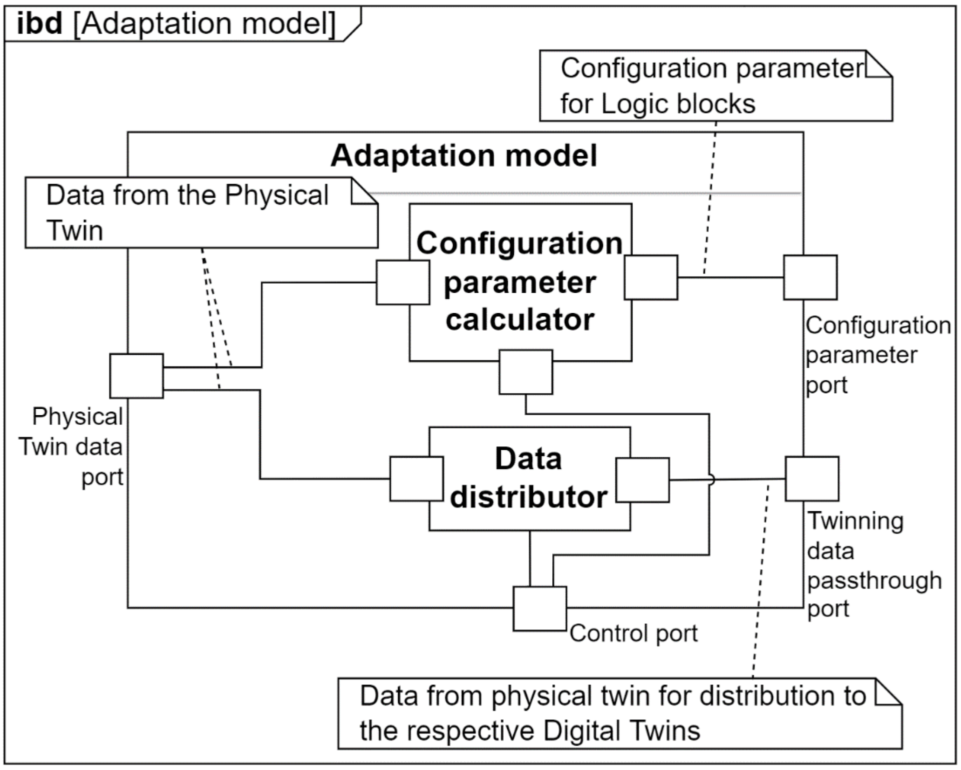
ibd [Adaptation model]
Configuration parameter for Logic blocks
Adaptation model
Data from the Physical Twin
Configuration parameter calculator
Configuration parameter port
Physical Twin data port
Data distributor
Twinning data passthrough port
Control port
Data from physical twin for distribution to the respective Digital Twins

Figure 26 Internal block diagram with connectors and blocks composing the Adaptation model.

When compared to the origin of the adaptation model in Figure 17, one can notice that not the input u and the DT output y are missing as inputs to the adaptation model as presented in Figure 26. The reason for the previous is that either it is part of the data from the Physical Twin, or a reference input is applied which can enable the omission of measurements of the input to which the data from the Physical Twin y_{PT} is the output. Thus, it does not need to be given as an input to the adaptation model. An example in the UUV UC would be a sequence of standardized tests with high explanatory power.
For the omission of the output y, there are multiple ways to calculate the configuration parameters such as physics-based, prior knowledge (like look-up tables) or data-driven approaches the configuration parameter calculator is simplified to just using the output data from the physical twin y_{PT} and outputting the configuration parameters. It is possible, especially in the data-driven approach that the Configuration parameter calculator triggers a simulation using the logics and sub-DTs w.r.t. the reference input and calibrates an error model compensating for misalignments between the simulation result and the physical twin data. This is assumed to be part of the Configuration parameter calculator and thus is not explicitly modelled.

In the training phase, the adaptation model can be simplified internally to a configuration parameter generator which is triggered by the DT controller as seen in Figure 27.

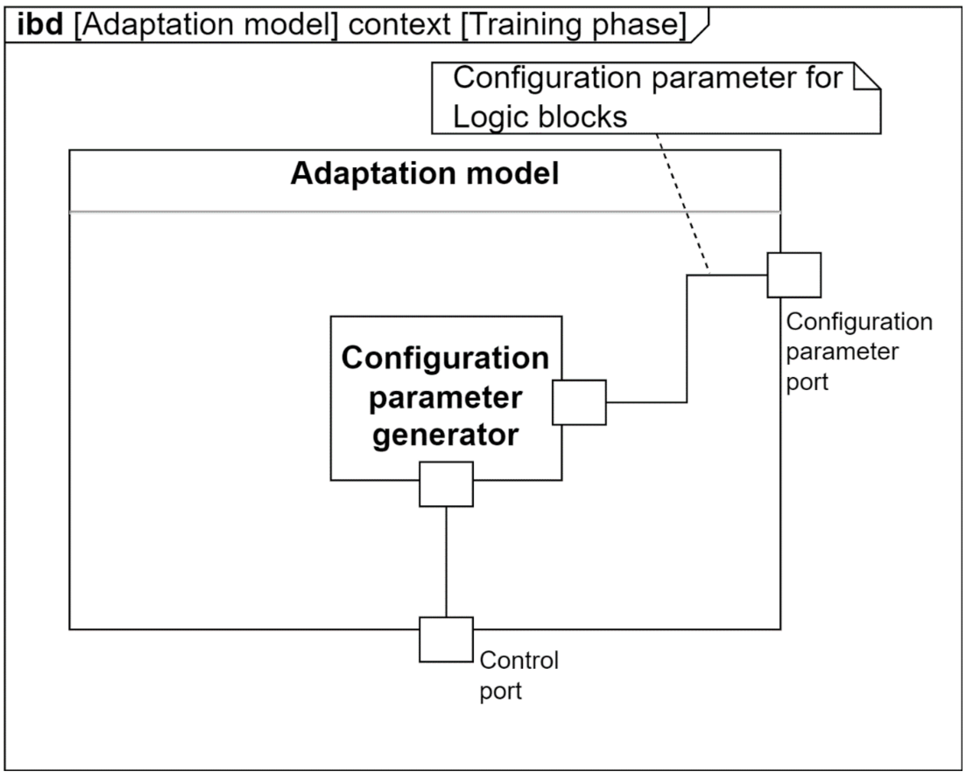
ibd [Adaptation model] context [Training phase]
Configuration parameter for Logic blocks
Adaptation model
Configuration parameter generator
Configuration parameter port
Control port

Figure 27 Internal block diagram of the adaptation model during the training phase.

3.3.4 Exemplary application

The architecture is developed with a use case from the highly automated vehicle testing domain that aims for automatic creation of a 3D-environment for vehicle tests. The system to be optimized is the testbed, including a virtual environment with a vehicle. For development of automated vehicle driving functions, critical scenarios are to be optimized to ensure testing time is used for the most insightful tests to find e.g., weaknesses in the functions. Also, this approach could be used as open exploration to build a database of scenarios. A scenario is usually characterized by specific criticality metrics that indicate its significance and potential impacts. The RLA proposes concrete scenarios with concrete parametrization/state values from logical scenarios with parameter ranges of state values which are executed on the virtual or real testbed, and the results are then computed to evaluate the scenarios.
Figure 28 showcases the logical view on the optimization system during the finetuning phase. Thanks to the reference architecture enabling hierarchical DTs, the DT of the vehicle test setup contains a virtual environment simulating e.g., the street as a logic block, and the DT of a vehicle enables a black-box integration of this sub-module. The vehicle DT can also contain other DTs as submodules e.g., sensors or chassis. A database is integrated to hold libraries of the virtual environment's assets. The simulation environment is connected to a vehicle its behavior it shall mimic. While in Figure 28 the fine-tuning phase is depicted, the only difference to the training phase is the substitution of the adaptation model by a parameter generator as seen in Figure 26 and Figure 27. This results in the minimization of necessary adjustments to the system including data formats as the variation generator and the twinning function output data in identical format.

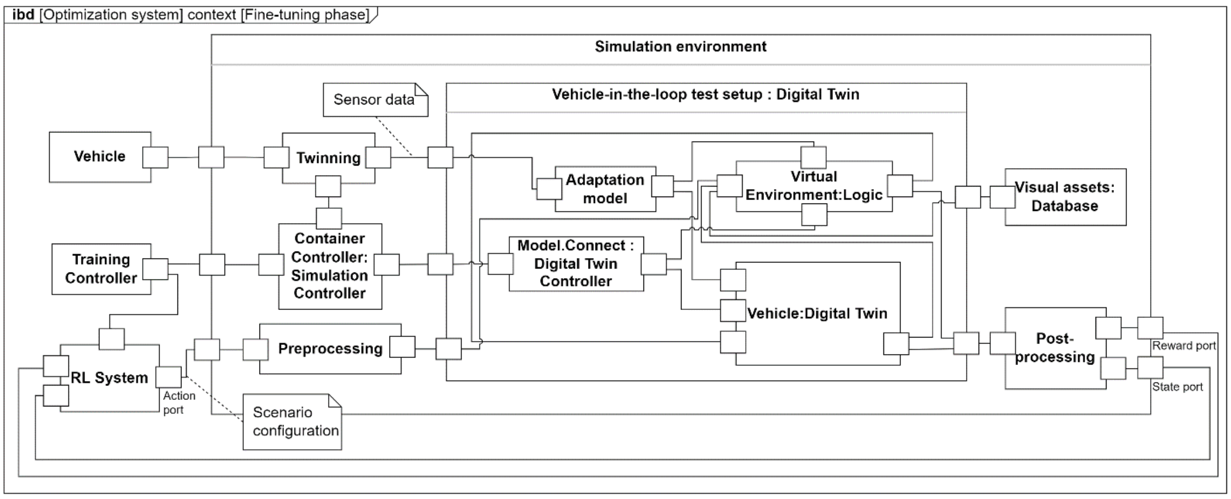

Figure 28 Exemplary application of the architecture to a vehicle-in-the-loop test setup during the operational and fine-tuning phase wherein scenarios are optimized, and additional data are gathered for further improvement of RLA.

3.4 Technical View

3.4.1 Introduction

Figure 16 shows how the various components and systems are connected. Through these connections, control signals and data can be exchanged, among other data. Interfaces, data flow specifications, data and metadata properties are particularly important in this context to ensure reliable communication.

Interfaces are of great importance because they enable access to the components and systems. They provide access to the outside world through which communication and data exchange between the components involved takes place. A well-designed interface facilitates the integration and interoperability of the components involved.

Data flow specifications describe how data is transferred between components. They ensure that data is processed in the right order, in the right form and at the right time. It is the foundation for reliable and efficient data processing. A well-designed data flow specification is a prerequisite for a correctly functioning and efficient process.

The properties of data and metadata are used to identify, classify and describe data. This information is essential for their use. In addition, they help to create an accurate replica of the physical object or system, and thus enable decisions regarding training and optimization.

Interfaces, data flow specifications, and data and metadata properties are critical to creating an architecture for a DT.

3.4.2 Interfaces between DTs for co-simulation

A DT is a virtual duplicate of a real object or system that makes it possible to optimize and analyze the behavior of the system in a simulated environment. Multiple DTs can interact in a co-simulation to replicate a complex system in a virtual environment. The interaction of DTs can take place in different ways. On the one hand, DTs can be coupled to each other, which means that they can influence each other. On the other hand, they can also coexist, so that several DTs perform their work simultaneously in a system. Furthermore, DTs can be organized hierarchically, with higher-level DTs containing lower-level DTs. An overview of the relationships between DTs can be found in the chapter Logical View. The communication of DTs used for co-simulation is done through interfaces. These interfaces specify which data, protocols and methods can be used for an interaction. With these specified interfaces it is possible to exchange data between different DTs.

Standardized communication is essential for an efficient and reliable process. Among other aspects, it enables a modular structure and thus facilitates the changeability of individual components. Depending on the requirements for transmission speed or reliability, different communication protocols can be used. For example, UDP is preferable to TCP if it is not necessary to ensure that a packet has arrived. This can increase the transfer speed.

Another aspect is the efficiency of data transmission. It is therefore important to know which data is transmitted, when and how between the components. Furthermore, it must be ensured that the data are not sent unnecessarily often and that only what is necessary is sent. In addition, there are the following methods with which the efficiency can be further increased: Data compression, caching, Publish/Subscribe models.

Due to the large number of components communicating with each other, errors and instabilities can occur. For example, incorrect settings can cause data to be generated incorrectly or corruptly. For this reason, it is important to implement techniques for analysis and error handling. The objective should be to quickly identify the source of the error.

The architecture of DTs can be very complex. One way to reduce this complexity is to divide the entire system into smaller units, as shown in Figure 16. This principle, which is called modularity, can help to make the system more understandable and maintainable. A further advantage of modularity is that the interfaces become smaller and thus more understandable.

Another way to further simplify interfaces is to divide them according to their responsibilities. This principle is also known as interface segregation (see [42]). In this case, an interface contains only the functionality that a user needs for his task. This avoids dependencies on something that is not needed and can help to reduce the compilation time of components when interfaces are changed.

When developing the interfaces, the above aspects should be considered to ensure reliable communication between the components. In addition, they facilitate the modification of the system when requirements change.

Over the past few years, different solutions for co-simulation have emerged in simulation and modeling, such as the Functional Mock-up Interface (FMI)[2], the High-Level Architecture (HLA) standard[3] and Model.CONNECT[4]. FMI is an open standard that can be used to exchange dynamic models. HLA provides an architecture for complex simulations and is a standard for distributed simulations. Model.CONNECT supports, among other things, simulations with multiple coupled models and enables the integration of real-time systems. These frameworks allow an easy connection between the co-simulation and the DTs. The use of these frameworks can reduce development time and thus lower development costs.

3.4.3 Performance view

In this section, the performance view of DTs in the context of AI-based training is discussed. The system's performance highly impacts the duration of the training process and thus directly impacts the development time and costs for the implementation. The focus in this section is on why performance is so important, what challenges and trade-offs exist that affect performance, and what methods can be used to improve performance.

As mentioned, the runtime speed of the DT has a great impact on AI-based training. In reinforcement learning, for example, an agent often requires a high number of iterations to learn a behavior. An iteration involves an action that is triggered by the RL agent. The action is then executed by the DT in a simulation environment. As a result of this action, the RL agent receives feedback in a form of reward or punishment, which it can use to adjust its behavior to make learning progress. The faster an iteration is, the faster the result is obtained.

From a technical perspective, various trade-offs can affect performance. For example, there may be a conflict between accuracy and execution efficiency. On the one hand, the high accuracy of models of a DT can lead to slow execution due to complex calculations. On the other hand, a DT's models may lose accuracy if optimization is focused only on execution speed. Another conflict exists between real-time capability and data processing. A DT's real-time capability depends largely on the processing speed. To generate output fast from input data, large data sets often need to be processed, which can affect execution speed, especially when hardware resources are limited.

Performance optimization must consider processing speed and data exchange and optimal use of hardware and software resources. The processing speed of DTs is crucial for fast and efficient simulations. The complexity of the models plays a central role here. Reducing model complexity can help to reduce computational effort and thus increase processing speed. However, this can also lead to a lower accuracy of the simulation results, which is why a compromise between model accuracy and execution efficiency must be found. This topic is addressed in more detail in Task 2.1. Another approach may be to develop surrogate models for a complex DT that can be evaluated quickly for training AI.

[2] https://fmi-standard.org/
[3] https://standards.ieee.org/ieee/1516/3744/
[4] https://www.avl.com/en/simulation-solutions/software-offering/simulation-tools-z/modelconnect

In a modular architecture, the DT consists of several components that communicate and exchange data. Efficient data exchange is important to achieve low response times and latency. This is especially important for time-critical applications. There are different approaches to optimize the data exchange. One approach is to reduce the amount of data that is transferred. In the simplest case, this can be achieved by transferring only the data that is required. In addition, the data can be compressed before it is sent to reduce the amount of data to be transmitted. Communication protocols such as MQTT (see [43]) can also help to transfer data more efficiently. In addition, the network over which a DT communicates can be designed in a way that bottlenecks can be avoided. Another approach is to prioritize the data to improve latency and response times.

The performance of a DT can also be improved on the software side by optimally utilizing the software resources. For example, threads can run on several CPU cores or on GPUs. This helps to distribute the computing load more efficiently. Especially with algorithms that can be parallelized easily, performance can be increased substantially with this method. For more information on the design of parallelized software, see [44] and [45]. Efficient data structures or caching techniques can also help to increase performance. Another alternative to increase performance could be the use of surrogate models which could be a faster alternative for slow DT (see section 4.3 on data-based modelling).

On the hardware side, the performance of a DT can also be accelerated by various measures.
For example, better load distribution can be achieved by additional hardware resources such as CPUs, GPUs or memory. However, it depends here on whether the corresponding software also uses these additional hardware resources. It is also conceivable to use faster hardware to increase performance. Another way to increase performance is to use the computing capacity of cloud services. Here, resources such as CPU and memory can be quickly adapted to the requirements. Cloud services like AWS and Azure offer different solutions and frameworks that can be used for a DT. For example, Azure offers a Digital Twin Definition Language (DTDL, [46]) that can be used to define models and DT. For certain tasks, it can make sense to use specialized hardware that has been developed for a specific purpose and can therefore be more efficient. This can be hardware for machine learning, for example.

In general, the modular component aggregation of the ASIMOV solution aids component-specific hardware allocation. Components can be distributed across different machines, leveraging components specific hardware specifications.

3.4.4 Information systems; internal and context and environment

Information systems (IS) can be important in the development and usage of DTs. Usually, IS consist of a software-based solution, such as application software and databases.
The main task of an IS is to collect, store, process, analyze, provide and link data. In the context of DTs, an IS can be used to collect data about a real-world object and create a DT from it. The structure of such an information system in the context of ASIMOV is discussed in the logical view chapter. An IS can also help facilitate interaction between the real object and its DT.

For a DT, the following data are relevant and can be managed by an IS:

- Internal data are information that can be determined directly from the real object, system, or process from which a DT can then be generated. This data can be obtained from various sources such as sensors, actuators, control devices, and integrated systems. For example, sensors can collect data such as depth information with Lidars or image information through cameras, while actuators and control devices provide information about movements, positions, and control commands. Internal data are important to accurately and completely map the behavior and performance of the real object to the DT.
- Context data refers to the information that defines the DT in terms of its circumstances, conditions or relationships to other objects or systems. Context can be determined by various factors, such as the location, the time, the intention of an experiment. The context is an important factor to be able to map the reality accurately.
- Environmental data refers to information about the physical environment and conditions that affect the real object that the DT represents. Environmental data allows the DT to better replicate the environment's effects on the behavior of the real object. Examples include temperature,

humidity, lighting conditions and weather conditions. It is difficult to draw a clear line between context and environment data, as they are closely linked. For example, environmental data depends on the context.

The consideration of internal, context, and environment data and the use of information systems makes it possible to create a DT that accurately represents the behavior of its physical counterpart and can enable realistic simulations, analyses and decisions.

In the context of DTs, the term digital thread is often used, which also has relevance for IS. DTs are based on data that comes from the digital thread. The Digital Thread evolved from product life cycle management (PLM), which captured the process from design to manufacture for physical products [47]. Unlike PLM, the digital thread extends the lifecycle record by integrating operations, maintenance and disposal. This extension makes it possible to gain a more comprehensive understanding of the entire product life cycle and further optimize product development.

One of the most important features of Digital Thread is a traceable, unique creation entry. This entry records data for each product, product group and all its components. In addition, the Digital Thread provides the ability to document interactions and transactions throughout the lifecycle. This is done throughout the entire lifecycle and extends to the final disposal of the product. A central aspect of the digital thread is the traceability of components, which relates to various elements. These include design iterations, manufacturing processes, tests, and quality measures.

Due to the comprehensiveness of the data, the Digital Thread can be a central element in the creation and updating of DTs. By providing a comprehensive and consistent set of data throughout the product's lifecycle, the Digital Thread enables the creation of an accurate and reliable DT. For more information on the topic, see [47].

3.4.5 Standards, data formats

In DTs, standards and data formats are of great importance. They ensure that different applications, systems and organizations can work together effectively by providing a set of rules and uniform data structures. Compliance with standards and the use of established data formats enables interoperability, reduces development effort and reduces the complexity of integrating different technologies. An organization that is committed to the development and promotion of standards and best practices is the Digital Twin Consortium [48].

The introduction of standards and data formats has several advantages:

- **Maintainability**: Facilitates the maintenance and updating of DTs, systems, and applications over time.
- **Interchangeability**: Individual components, systems or applications can be exchanged more easily.
- **Simplicity**: Reduces the complexity and effort of integrating different technologies and data sources.
- **Developability**: The use of standards and data formats leads to reduced development time for DTs, due to less time spent on customizing and integrating systems.
- **Interoperability**: Standards and data formats enable reliable communication between DTs, systems and applications and ensure reliable data exchange even between models that originate from different sources.
- **Reusability:** Standards enable the reusability of components, such as models.

Standards and data formats can therefore help to manage the complexity of the various technologies. However, several challenges can arise when selecting standards and data formats. Some of the most important challenges are:

- **Technology stack in the organization**: During development, components are often integrated into the existing technology stack rather than starting from scratch. This can lead to problems if the new technology is not compatible with the existing one.
- **Number of standards and data formats:** There are many standards and data formats, which have their advantages and disadvantages and are sometimes sector or application specific.

Choosing an appropriate standard or data format can be difficult given the number and their differences.

- **Interoperability**: A standard or data format that enables interoperability with other DTs, systems, applications and organizations can be challenging to choose. This requires careful evaluation of the compatibility of different technologies.
- **Sector-specific requirements**: It can be difficult to find a common standard or data format that is suitable for all stakeholders, especially if they come from different sectors and have different requirements.
- **Acceptance and dissemination**: A standard or data format that is not widely used is difficult to establish. Its use therefore depends on its acceptance and dissemination.
- **Technological change**: Advances in technology may lead to new standards and data formats, while older ones may become obsolete. Choosing a standard or data format that can adapt to technological changes while being backward compatible can be a challenge.

Facing these challenges requires careful consideration of which standards and data formats best fit the specific requirements of a DT. For challenges and solutions related to standardized software patterns, see also [49].

In this context, it is important to look at standards that have been developed specifically for DTs and those that are used by DTs but have not been explicitly designed for them.
Different organizations and committees are involved in the development of standards for DTs. These organizations include the International Organization for Standardization (ISO), which is involved in developing standards for a wide range of industries and technologies. ISO standardizes processes, data formats and interfaces relevant to DTs. Also relevant is the International Electrotechnical Commission (IEC), which develops electronic technology standards. In addition to these organizations, there are other initiatives and committees involved in developing standards for DTs.

General standards not explicitly developed for DTs can be an important building block. These standards include areas such as communications, security, and data formats. However, there are standards developed specifically for DTs to ensure their standardized implementation and compatibility. ISO 23247 is an important standard that addresses the topics concerning DTs.

The ISO 23247 series provides a framework to support the creation of DTs in manufacturing. It refers to various elements such as personnel, equipment, materials, processes, facilities, environment and products. The series includes general principles and requirements for the development of DTs, a reference architecture with functional views, a representation of manufacturing elements, and requirements for information exchange. For more details, refer to [50].

In addition, ISO and IEC have a Join Technical Committee with the primary objective of developing standards for the Internet of Things (IoT) and DTs and related technologies, called ISO/IEC JTC 1/SC 41. For more information on the topic, see [51].

The choice of data formats for DTs depends on the task they are intended to accomplish and the area in which they are used. The following file formats can be relevant for the creation and use of DTs:

- **Data structure formats**: These formats are used to organize data in a structured form, which can then be used for data exchange, for example. Common formats are for example JSON and XML.
- **3D modeling formats**: These formats are often used in areas where a three-dimensional visual model needs to be created. Common formats are, for example, FBX and OBJ. Special CAD formats can also be used for detailed design information.
- **Other formats:** Specific formats may be required depending on various factors such as requirements. For example, GIS formats that represent geographical data can be used for the environment in which the DT is embedded.

It is important to note that DTs are often not limited to single data formats, but may use a combination of formats to provide, for example, a complete and accurate representation of the physical object or system.

3.4.6 Deployment, Software aspects

Deployment generally refers to the process of delivering software to a target environment. It includes the compilation and testing of the software and its installation and configuration in the intended target environment. Deployment can be a very complex task, with various software and hardware aspects to consider. In this section, we will look at these issues in more detail to gain a deeper understanding of the deployment process and its importance when using DTs.

Regarding a suitable target environment, several challenges arise that are essential for the deployment. For example, the costs caused by a particular deployment strategy must be considered. There are immediate costs for hardware, software and personnel and long-term costs for maintenance, upgrades and downtime. In addition, the short- and long-term viability must also be analyzed. For example, the cloud's use enables significant cost savings, as only the resources requested are charged. This means that the infrastructure can be utilized optimally.
The protection of confidentiality and intellectual property is another important aspect and must also be considered when deploying the software. Security measures and data protection practices must ensure that the risk of data loss is minimized. This is particularly important for sensitive data or intellectual property.
Another aspect relates to the system to be deployed and whether it is critical to the success of the task. A failure of the system could potentially put the company's goals at risk. Therefore, reliability is another aspect that needs to be considered.

Various software aspects that must be considered during the development of the DT contribute to its successful deployment and operation in the target environment. For example, aspects such as modularity, scalability, performance, security, compatibility, and maintainability can improve the performance and reliability of the system. For this reason, it should be known in advance in which environment the DT is to be deployed.

As mentioned earlier, the choice of a deployment strategy can depend on many factors. There are several options on which hardware the deployment can take place. The advantages and disadvantages of some of these options are discussed in more detail below.

In local deployment, also known as on-premises deployment, the DT is installed and operated on the company's hardware. The advantage of this type of deployment is the high degree of control over the hardware and software. Among other benefits, this can lead to higher security, for example when the system is operated in an isolated network, and to higher performance, as the system can be tailored to the specific requirements. Direct access to the hardware makes it easier to make changes. However, the disadvantage is the higher cost of hardware, maintenance and personnel.

Cloud-based deployment, on the other hand, uses the resources of cloud service providers, who make their computing capacity and storage space available to their customers. Cloud service providers offer various cloud service models, including Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Depending on the model selected, the overhead of system maintenance and the degree of control over the system can vary greatly. A key advantage of cloud-based deployment is that resources can be added and removed easily. This makes it possible to scale the application and use resources on demand more easily, which can significantly reduce costs. On the downside, there are risks related to data protection and security, as well as access to the cloud service from the Internet if there is no reliable Internet connection.

With edge computing, data is processed close to the data source instead of being sent to a distant data center or the cloud for processing. Processing usually takes place on the device itself or nearby, e.g., on a local server. This can reduce response time and bandwidth usage since it avoids sending unnecessary data to a central location. Transferring only necessary information and avoiding deep system access can improve security. However, there are also disadvantages to consider. The high level of distribution may increase security risks. The cost of more complex implementation, management and maintenance compared to centralized solution can be higher.

The implementation of the “Twinning” block (see figure 22) is concerned with such considerations. In use cases with a small number of even just one system to be optimized, the retrieval of data directly from the system’s sensor to the adaptation of DTs in the simulation environment may be feasible, but at scale and in distributed system, this runs into problems fast. Thus, a framework is required to orchestrate the retrieval and pre-processing of data at or close to the CPS. An example for this BaSys 4, a reference architecture with its open-source reference implementation BaSyx [52]. The central idea of this framework is the enveloping of the real system in an “asset administration shell” (AAS) (see Figure 29) which orchestrate the retrieval of system data from the real device and storing and computing it in sub-models as well as providing a standardized and simple interface for communication. High-level applications (as the DTs’ adaptation models in the ASIMOV solution) can then “control, optimize and monitor the digital factory” or any system [53]. This can basically be seen as the interface for physical-digital twinning and vice-versa or “Twinning” block. The communication would be over network enabling the RLA being part of the CPS or somewhere in the cloud.

While the AAS can govern sub-modules which can also be connected to simulation models (see Figure 29), the RLA does not interact with these. While their usage fundamentally would be able to reproduce the behavior necessary for the training of the optimization RLA, in case they meet the requirement on accuracy, there are benefits to the integration of the models into a dedicated environment simulation as response times and scalability are easier to manage. A direct connection of the RLA with the submodels via an AAS is not replicated in the above shown architecture. Another possibility to combine both frameworks is that the Basys sub-modules take over the task of the adaptation model and only model configurations are transferred via the AAS to the DTs in the ASIMOV framework. However, this defies the self-containedness including its advantages of the DT as proposed in the ASIMOV DT architecture.

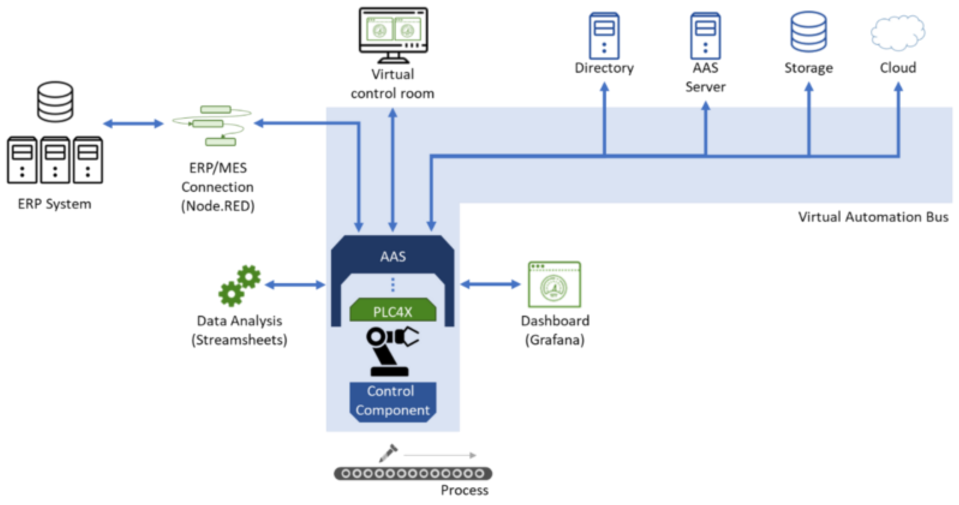

Figure 29 Overview on the high-level architecture of Basyx. The ASIMOV solution would be situated alongside the virtual control room, interacting with the system via the AAS. Source: [54].

The choice of the right deployment strategy depends on various criteria, such as data protection guidelines, specific requirements of the DT, the existing infrastructure and the budget.

For a more detailed discussion and in-depth information on deployment, please refer to [55]. There you will find a comprehensive coverage of various aspects of deployment, including detailed descriptions and use cases.

3.4.7 Security and trustworthiness

In today's connected digital world, security and trustworthiness are critical. Security refers to protection against unauthorized access, data loss and manipulation and ensures the integrity and availability of information. Trustworthiness, on the other hand, refers to a system or information's reliability and credibility.

Security measures should be built into the development and deployment process of a digital twin from the beginning. This section provides a brief overview of the importance and necessity of security measures. The findings on security and trustworthiness are based on the work of the Digital Twin Consortium, which can be read in detail in [56]. The Consortium recommends considering the following key aspects for security and trustworthiness:

- **Data Encryption:** Data encryption ensures the confidentiality of data before it is stored or transmitted.
- **Device Security:** Controlling access to device data is necessary to prevent unauthorized access and manipulation. This can be achieved by creating suitable privileges and defining a framework for enforcement.
- **Security:** The security measures protect the digital twins and their data. The measures are intended to prevent unwanted or unauthorized access, changes or destruction. This ensures confidentiality, integrity and availability.
- **Privacy:** Privacy gives users of digital twins control over the collection, storage and sharing of their data to protect their rights and privacy.
- **Safety:** The purpose of safety measures is to ensure that the digital twins are operated without posing significant risks to people, property, or the environment.
- **Reliability:** The reliability of a digital twin refers to its consistent performance, quality of service, availability and accuracy. This ensures consistent and accurate results.
- **Resilience:** The digital twin's resilience ensures acceptable performance levels even during disruptions. The goal is to maintain continuous functionality despite interruptions, failures, or attacks.

4 Digital Twin Behaviour

The concept of Digital Twin (DT) has emerged here to bridge the virtual and physical worlds, to optimize the operation and to enhance decision-making processes. Behavior modeling is a crucial aspect essential for the development and effectiveness of DTs. This section looks at the requirements for behavior modeling in this context and explains the methodologies.

4.1 System Behavior

Various definitions of system behavior are delivered in literature, each of them aims at exploring different categories of system specifications. The Systems Engineering body of Knowledge (SEBoK) defines behavior as “the effect produced when an instance of a complex system or organism is used in its operational environment”.
The behavior of the system is anticipated as event(s) that triggers other event(s) in that system or its environment. The event refers to a change in one or more structural properties of the system or its environment over time. The predecessors and/or consequences of the events can be used to study the system behavior [57]. In the case of systems with higher level complexity, e.g., CPS‘s, these events are not fully observable and moreover the changes due to these events are not necessarily deterministic.

State Transition Model. Describing the system behavior using a sequence of states and transitions between those states is one of the dominant behavior modeling paradigms [58]. The states comprise the system’s parameters and the behavior specify the evolution in values of those parameters. The causal explanations for the state transitions include physical laws, mathematical equations, functions of the subsystems, structural constraints, or other behaviors. The state space in this paradigm is either continuous or discrete. For a continuous representation, we assume that the system state changes continuously over time according to a set of differential equations. By discrete state space modelling, the behavior of an operating system is represented using a finite set of discrete state variables over discrete time steps. Within this paradigm, we assume that between two consecutive time steps, the system remains unchanged. At any time, the system's state vector is constituted by valuing state variables.

The dynamical system can be modelled by continuous state space as $\frac{d}{dt}x = f(x, t, u; \beta) + d$ and $y = g(x, t) + n$, where $\frac{d}{dt}$ represents the differential operator, x is state of dynamic elements, f is state function while the dynamics change continuously over time, t represents time, u input, β indicates parameters, d disturbance, y observable outputs, g output function and n represents noise. In many instances we do not have knowledge about the entire set of disturbance and noise elements. This can be conquered by feeding the generated output by the CPS of interest into DT.
Note that in a probabilistic transition modelling, we assess the behavior of the system as a transition model $P(x, a, x')$, i.e., the probability of reaching state x' from state x applying the action a .

The behavior of a system often goes beyond the aggregation of its individual components that make up the system. Consider the example of double pendulums, where the behavior cannot be replicated by linearly combining two single pendulum models. Complex systems frequently exhibit emergent behaviors where the properties of the system surpass those of its individual parts. The prediction and analysis of emergent behaviors, which may lead to undesirable outcomes, is one of the challenges in the development and modeling of such complex systems [59]. The uncertainties and non-linearities of the actual system that are not (or cannot be) modeled need to be explored at the level of model's behavior and overall performance.
We can expand the boundaries of integrated component models by modeling the behavior of a unified system and extending the horizon of behavioral patterns created by the underlying structure. This expansion can be achieved through capturing cause-and-effect relationships. The behavior of a CPS depends on its individual components, their interaction and the inputs from the CPS environment. The goal is to replicate the behavior of CPS by the DT sufficiently well, regardless of the existing uncertainties inherent in CPS and its environment modeling.

4.2 DT Behavior

A DT is characterized by its ability to interface the actual system and exchange data and control information. By tracking the past and current behavior, the DT predicts the future behavior and thus provides real-time decision information to optimize the future operations. Like other software models, by developing formal DT specifications that capture potential behavior and using model checking techniques, we can accurately determine if specific requirements are met. In instances where requirements are not met, the counterexample traces allow us to diagnose faulty behaviors. Here, the focus is on investigating behavioral requirements within a scenario-based framework, thereby exploring system performance in predefined scenarios.

Figure 30 illustrates a fundamental connection between the CPS and the DT. Both systems receive identical inputs, with the DT additionally taking input from the CPS's output.

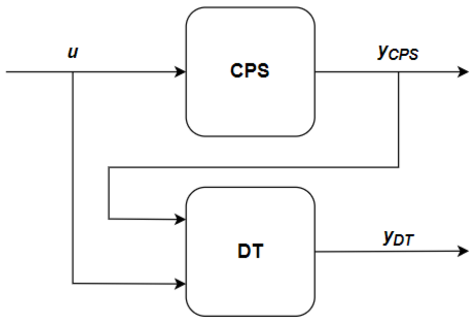

Figure 30 Transferring data between CPS and DT.

To characterize the CPS's behavior at discrete time steps, denoted as $i \in N$, we evaluate its transition to state $x_{CPS}[i]$ resulting in output $y_{CPS}[i]$, after being stimulated by input u at the state $x_{CPS}[i-1]$. This can be represented by the conditional probabilities:

$$P(x_{CPS}[i]|x_{CPS}[i-1], u[i]),$$
$$P(y_{CPS}[i]|x_{CPS}[i]).$$

The development of the DT should aim to replicate a similar behavior, ensuring it generates transitions and outputs like the CPS. Thus, the DT's conditional probabilities are expressed as,

$$P(x_{DT}[i]|x_{DT}[i-1], u[i], y_{CPS}[i]),$$
$$P(y_{DT}[i]|x_{DT}[i]).$$

Assuming functional equivalence between the CPS and DT implies that when stimulated with the same input from identical initial states, they should exhibit analogous transitions and yield equivalent outputs.

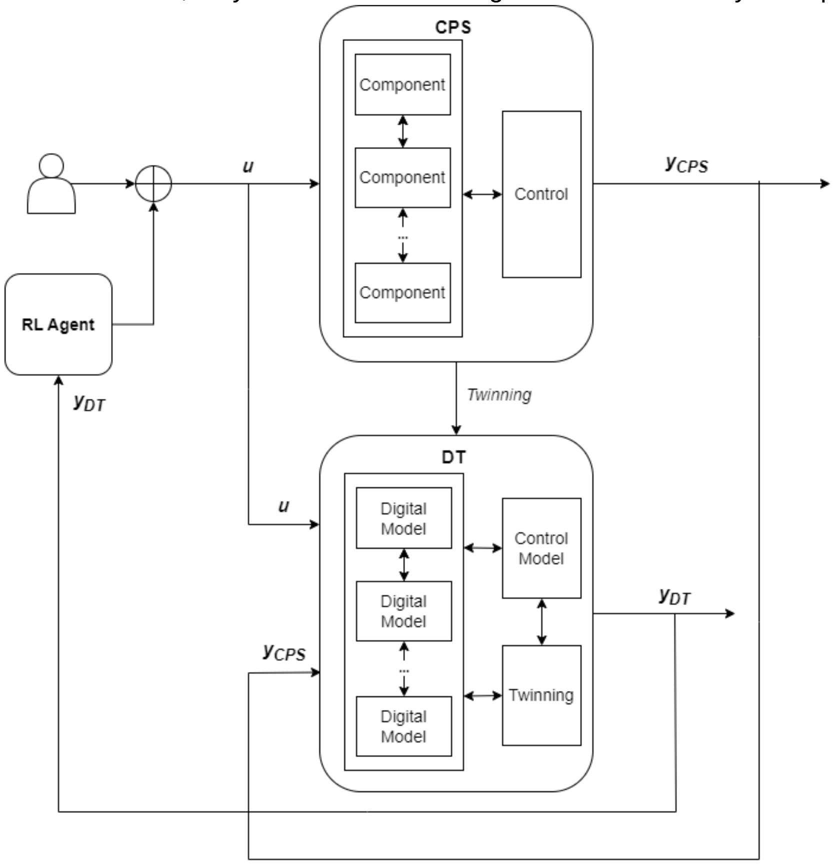

Figure 31 Overall architectural view of exchanging data between CPS, DT and RL Agent in ASIMOV for the purpose of behavior modeling

The DT in ASIMOV operates within an environment comprising the RL Agent and the CPS, often referred to as the Physical Twin (see Figure 16).

In Figure 32, the detailed representations include $\boldsymbol{u}$ denoting the set of known (controllable) system inputs, $\boldsymbol{y_{CPS}}$ denoting the observable outputs of CPS, and $\boldsymbol{y_{DT}}$ denoting the data generated by the DT. Notably, the models of system's components employ different functions, subfunctions and algorithms. The *Twinning* data originates from the operational CPS, i.e. sourced from its environment, communication medium or the CPS itself. This data is usually available as time-series data, primarily scalar data capturing the values of sensors and actuators. The effects of DT on its environment can be characterized by its role in providing training data for the RL Agent, thereby shaping the learned policy, and consequently influencing the CPS optimization.

The creation of the DT involves initially crafting digital models by either learning the physics-based models of CPS components or utilizing historical data gathered from a representative CPS. The next steps entail integrating real-time data generated by the operational CPS to refine the initially trained DT. These iterative processes prepare the DT to capture the known behaviors of its evolving CPS counterpart.

Evaluating the behavior of the DT necessitates investigation across different phases of ASIMOV.

Various modeling techniques for system behavior can be considered in different phases, including:

- Modeling based on control theory, using laws of physics and mathematical models;
- Data-driven modeling using machine learning techniques;
- Hybrid modeling, which integrates both control and data-based modeling;
- Alternative approaches such as graph-based frameworks and ontology-based models.

Throughout the DT development phase, the selection of the most appropriate approach depends on the availability of conceptual knowledge and/or collected data from the CPS use case. The resulting behavior model undergoes further refinement in developmental phases.

4.2.1 Sequential Behavior

Along with the definitions of system behavior, the behavioral architecture is "an arrangement of functions and their subfunctions and interfaces (internal and external) that defines the execution sequencing, the conditions for control or dataflow and the performance requirements to satisfy the requirements baseline" [modification of ISO/IEC/IEEE 24748-4 for behavioral aspects by SEBoK]. The behavioral architecture of a system describes what is happening inside that system to generate the output, given the inputs and the control configurations. Segovia et al. [60] indicate the necessity of evolving architecture in order to integrate system changes, such as new components, interfaces or connections to other components, or to adapt the system behavior.

The process interactions arranged in time sequences are summarized and shown in Figure 33 and Figure 34. Sequence diagrams are generally used to represent the interactions between objects in the sequential order that the interactions take place. These diagrams are useful for documenting how the system model should behave. We aim to establish the transition from the requirements formulated for the use cases to the next formal level of refinement, e.g., the DT behavioral architecture in one or more sequence diagrams. Each element in these figures is drawn by bigger rectangles and the attached dashed line to represent the lifeline, e.g., CPS, DT or Control and Logic. The vertical axis represents time proceedings down the figure. Along the lifelines, the thin rectangles depict the period of activation, and the smaller rectangles indicate the self-messages that invoke operation in the corresponding lifeline. The vertical dimension of a sequence diagram shows the time sequence of calls as they occur, top to bottom, and the horizontal dimension shows the object that sends message or information to the object that receives it, left to right. The self-arrows are used to show the processes that occur within the system.

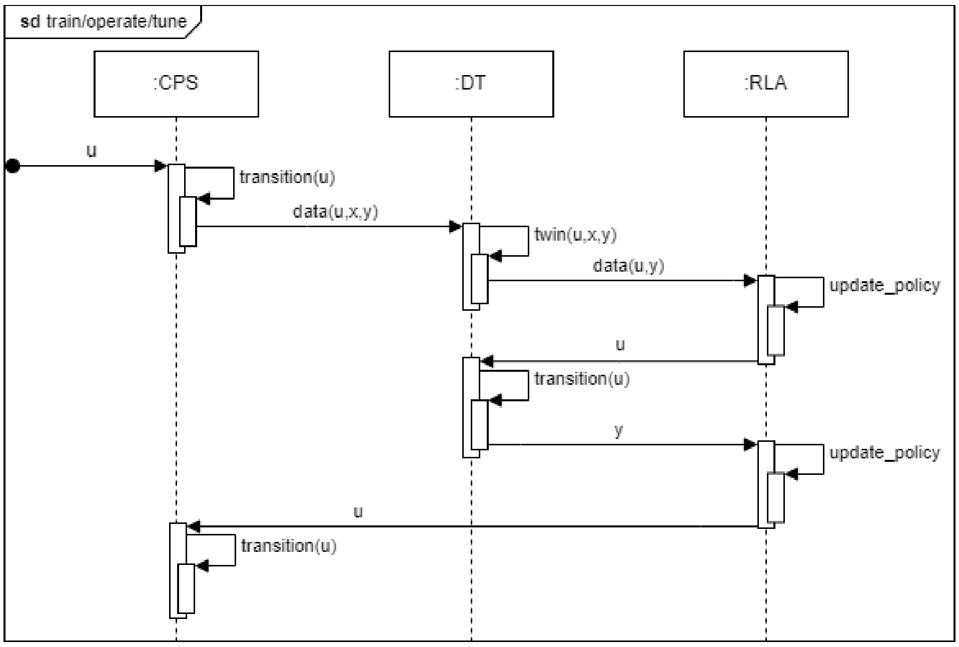

Figure 33 Sequence diagram of interaction between CPS, DT and RL Agent.

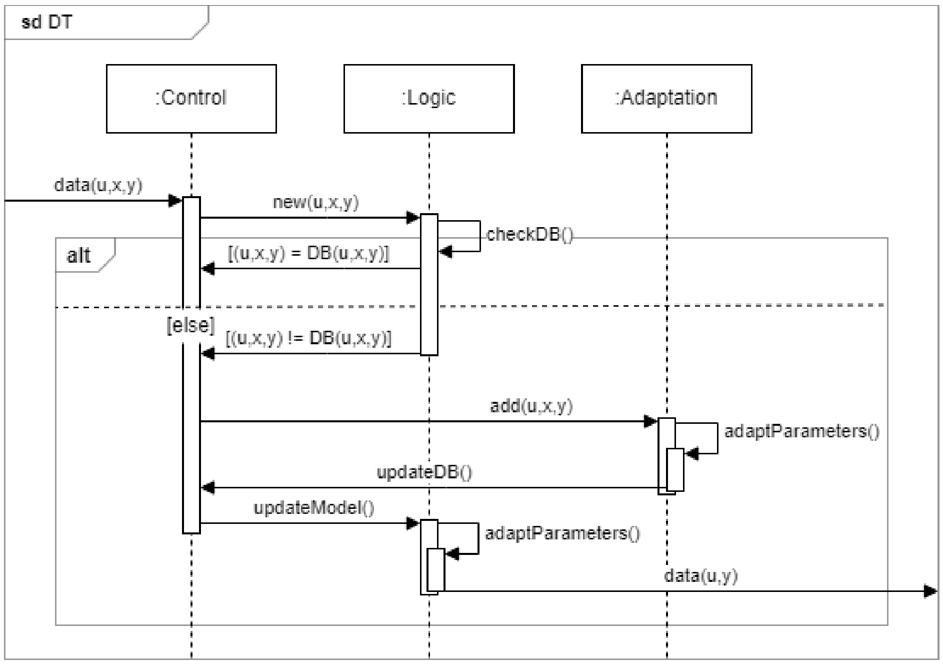

Figure 34 Sequence diagram of DT components.

Often, the system of interest incorporates several connected components that process sequentially or simultaneously to produce the system's final output. The system is triggered by an external input which is fed into a subset of these components while the rest of components receive the generated outputs by the first subset as inputs. To differentiate the external inputs from the intermediate ones for DT, [61] defined a set of stimuli S using the set of inputs of all incorporated components in CPS but excluding the ones that are shared with the outputs set. In other words, $S \coloneqq \{s \in U_{DT} | s \in U_{CPS} \land s \notin Y_{CPS} |\}$. Feeding DT with the identified stimuli, the same states should be reached in DT as in CPS, which the authors call "state replication".

Often, the system of interest incorporates interconnected components that operate sequentially or concurrently to generate the system's final output. The system is triggered by an external input which is fed into a subset of these components while the rest of components receive the generated outputs by the first subset as inputs. To distinguish between the external inputs from the intermediate ones for DT, [61] defined a set of stimuli S using the set of inputs of all incorporated components in CPS but excluding the ones that are shared with the outputs set. In other words, $S := \{s \in U_{DT} | s \in U_{CPS} \wedge s \notin Y_{CPS}|\}$. Feeding DT with the identified stimuli, the same states should be reached in DT as in CPS, which the authors call "state replication".

4.2.2 Adaptive Behavior

In response to deviations in the established patterns of CPS behavior that lead to deviations from DT predictions, the DT should go through adaptation. For instance, the behavior of the test bed of the UUV use case can be influenced by changes in temperature or humidity level of the room. They necessitate corresponding adjustments in DT behavior to maintain prediction accuracy. The behavioral adaptations refer to instantaneous changes and those developed over time to fulfill the DT objectives (including training data for the RL Agent in ASIMOV).

A primary strategy for adapting DT behavior involves modulation of various parameters. This approach relies on parameterized models. The adaptation in this case is realized by adjusting the parameter values. The data from the CPS, including the sensor data and actuator values collected from physical components, serve to update DT, preparing it for further prediction. Figure 34, for example, illustrates a sequential behavior of DT when presented with *data(u,x,y)* collected from CPS. As earlier described '*u*' represents input, '*x*' represents state and '*y*' represents the observable output. This data undergoes a comparison with the known behavioral data within the DT database by the logic component. If the data is not listed, it will be incorporated, and the DT will be adapted accordingly to enable it to generate similar data(u,x,y) in the future. This adaptation involves parameter adjusting and potential model refinements.

However, due to multi-scale nature of the integrated components of CPS, it is not always a straightforward task to identify the parameters and learn the models to map the changes of CPS into DT. Statistical and machine learning techniques by use of data, deliver key insights about the system and subsequently enable us to model the system behavior. For instance, [62] proposed a Mixture of Experts using Gaussian process (MoE-GP) approach to learn the time-evolution of different system parameters, which are in this case selected by the physics-based insight. The identified parameters evolve at different time scales and each of the experts within MoE framework learns the evolution of the system parameter at a single scale using GP. The observable outputs, y_t, at discrete time points are assumed to be generated from M independent hidden states, $x_t^{(m)}$with $m = 1,2, \dots, M$ which are referred to as experts. The evolution of hidden states is assumed to be a Gaussian process, and to obtain the observed data y_t, they couple the hidden states in a generative way. Once the hyperparameters and coefficients are estimated, the behavior of the evolving system can be predicted.

The data retrieved from the CPS, including the sensor data and actuators value collected from its physical components, serve as the foundation to update the DT preparing it to predict future states. However, due to multi-scale dynamics of the integrated CPS components, identifying the parameters and learning to map CPS changes is not always straightforward. For instance, the components often operate at different scales and exhibit different aging characteristics. The tires in a vehicle experience wear and tear at a different rate compared to other components. Several factors can influence their aging, such as road conditions and driving habits. On the other hand, the trees in the driving environment have their own aging processes impacted by different factors such as weather conditions and seasonal variation.

Statistical and machine learning techniques, exploiting available data, deliver key insights into the system and facilitate the modeling of its behavior. For example, [62] proposed a Mixture of Experts using Gaussian process (MoE-GP) approach to learn the time-evolution of various system parameters. The parameters, e.g. selected based on the physics-based insights, evolve at different time scales. Within the MoE framework, each expert learns the evolution of the system parameter at a single scale using GP.

The observable outputs, y_t, at discrete time points are assumed to originate from M independent hidden states, $x_t^{(m)}$with $m = 1,2, \dots, M$, referred to as experts. The evolution of hidden states is assumed to be a Gaussian process, and to obtain the observed data y_t, the hidden states are coupled in a generative manner. Once the hyperparameters and coefficients are estimated, the prediction about the behavior of the evolving system can be made.

In a broader context, the adaptation can also take the form of structural adaptation and behavioral adjustments. The structural adaptation aims to adapt the system behavior by changing its architecture, while the behavioral adaptation aims to modify the functionalities of computational entities.

PobSAM (Policy-based Self-Adaptive Model) offers a policy-based modeling framework for the self-adaptive systems, supporting behavioral adaptation [63]. This model consists of three layers, namely actor, view and manager layers. The functional behavior is implemented by computational components in the actor layer. The view layer addresses the actual state of actors. The manager level consists of the managers, each equipped with different configurations (policies), with only one active configuration at a time. Guided by the active configuration, a manager governs the behavior of its corresponding actors. HPobSAM (Hierarchical PobSAM), an extension of PobSAM, supports modeling large-scale adaptive systems. The self-adaptive modules have been integrated into PobSAM as a structuring feature [64].

4.2.3 Validation of DT Behavior

One of the ultimate goals of behavior modeling is to provide models that integrate the whole system behavior at a level of abstraction that enables its validation during DT development and also supports traceability of the behavior and its validation across the operation and fine-tuning processes. We are searching to specify the appropriate states, such that the resulting behavior realized by applying those states, is qualified to meet requirements. These requirements cover the behavior space, including critical behavior, and the capability to be scalable/reusable for different functional specifications.

The state variables are mostly selected in accordance with the system equation-based models and following methods of system identification. The state variables evolve over time depending on their current values and on the externally imposed values or input variables. If we have the initial state (the state vector of system valued at initial time), along with the inputs over time interval from that initial point onwards, we will be able to determine the state after that time interval. The other point to consider in selecting state variables is balancing controllability (on which parameters the input *u* has most impacts) and observability (which parameters can cause more changes on measurable or observable *y*).

Regression analysis of observed data is a robust statistical modeling technique for elucidating different variables that contribute to the system behavior and estimating the relationships among them.
Structural Equation Modeling (SEM) and Factor Analysis (FA) represent two convenient instances of regression-based techniques to model DT behavior by exploring the measured and latent variables while exploring their interdependencies. By definition, a latent variable is the variable that can only be indirectly inferred, while an observable variable is directly observed or measured. By definition, a latent variable is the variable that can only be inferred indirectly while an observable variable can be directly observed or measured. In this way, a factor describes and explains a set of observed variables that have similar response patterns. Moreover, a factor is a set of observed variables that have similar response patterns, and they are associated with a hidden variable.
The conceptual knowledge about the system is then beneficial to supervise the learned model by including or excluding the dependencies between the variables.

4.3 Qualitative Modeling and Behavior Types

Often in investigating a complex system, we face situations where the detailed (or deep) models of the underlying mechanisms, particularly continuous aspects, are not accessible, or where the state variables are not directly observable. In such cases, the qualitative modeling of the system begins with description of the known structures within that system for reasoning. This often involves drawing a directed graph from the initial state to the possible future states, indicating the relation between these states [65]. The

possible behaviors of the system are then represented by different paths within the resulting graph, starting from the initial (current) space.
The qualitative models are useful precise knowledge is lacking and allow reasoning at a higher abstract level. The goal of creating qualitative simulation models is to capture the essential characteristics of the structure and the behavior of the system. For instance, to understand the dynamic behavior of CPS and relations between various factors, semi-structures or unstructured approaches can be employed for extensive data collection, such as conducting interviews with human experts.
An exemplary qualitative model, as depicted in Figure 35, represents the influence between three parameters, temperature, wearing and sensor error rate, for the UUV use case. The operation of the test bed normally leads to wear and tear in some of the machinery components as well as an increase in the temperature increasement. These two parameters have mutual influence on the frequency of measurements error of the implemented sensors on the test bed.
Assuming equation-based model or observable data are not available, by following a qualitative approach, we can bring the (known) uncontrollable and unobservable elements of CPS, including the inputs from the external environment, into the DT and replicate the influence by DT behavior.

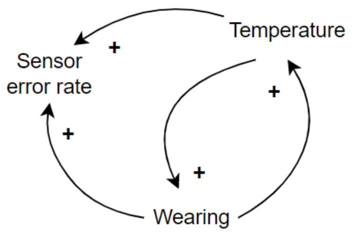

Figure 35 Exemplary influence diagram.

Every system of interest can have behavior traits that define how it responds to various types of inputs. Acquiring the knowledge about these traits, or types of behavior, and transferring it into DT will result in an effective prediction of the behavior of the CPS over time. Depending on the system specifications, various behavior types, e.g., emerging behavior, critical behavior or challenging behavior, are required to synthesize the system's triggered collective responses.

4.4 Data-based modelling

4.4.1 Introduction

Introduction 1.2.2 describes modelling challenges which arise in many disciplines and scientific fields. As explained in 3.3.3, the DT can consist of components which are DTs or models themselves and arise from different development activities. We will use DT, Digital Model and model here interchangeably. Possible approaches of model building are

- Modelling primarily based on domain knowledge, first principles
- Modeling using field data (e.g., examples of observed input-output)

If much domain knowledge is present, it may be possible to "write down" the model from first principles alone. Often, some numerical constants (e.g., mechanical or electrical properties) may still be unknown. In previous sections, these are the **configuration parameters.** At this point, we focus on the case of field data. It is still beneficial to use as much domain knowledge as possible, e.g. a causal interpretation can help deal with the different roles of the variables, and physical insight helps in formulating the exact form models. Data is processed into the appropriate level of aggregation and summarized in a convenient way, a.k.a. feature engineering in machine learning contexts. In this situation, model building is based in part on data. In statistics [66], the jargon is that the model parameters or coefficients are estimated; in machine learning the model is trained and weights are found, in other domains the model's numerical constants may be "calibrated" to data. Here the model parameters, coefficients, weights, calibrated

constants coincide with the configuration parameters. A further link is the output function $y = g(x, t)$ from 4.1. We switch to the generic notation $y = g(x_1, \dots, x_p)$ with inputs, or "predictors" in statistics, and time variable t is then one of the predictors.
Figure 36 is one possible taxonomy of data-based modelling in this situation.

4.4.2 Taxonomy of data-based modelling

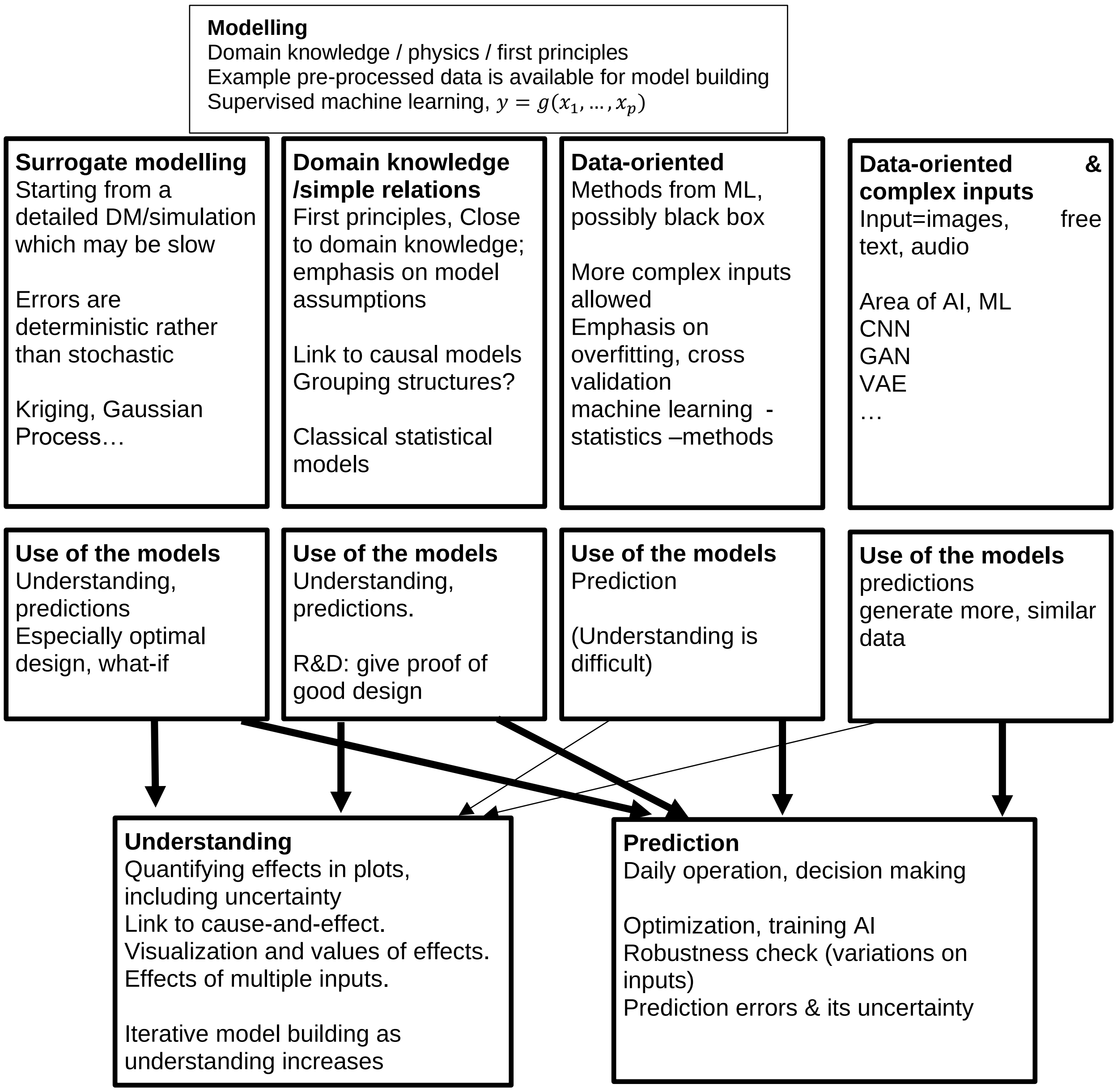
Modelling
Domain knowledge / physics / first principles
Example pre-processed data is available for model building
Supervised machine learning, $y = g(x_1, \dots, x_p)$
Surrogate modelling
Starting from a detailed DM/simulation which may be slow
Errors are deterministic rather than stochastic
Kriging, Gaussian Process…
Domain knowledge /simple relations
First principles, Close to domain knowledge; emphasis on model assumptions
Link to causal models
Grouping structures?
Classical statistical models
Data-oriented
Methods from ML, possibly black box
More complex inputs allowed
Emphasis on overfitting, cross validation
machine learning - statistics –methods
Data-oriented & complex inputs
Input=images, free text, audio
Area of AI, ML
CNN
GAN
VAE
…
Use of the models
Understanding, predictions
Especially optimal design, what-if
Use of the models
Understanding, predictions.
R&D: give proof of good design
Use of the models
Prediction
(Understanding is difficult)
Use of the models
predictions
generate more, similar data
Understanding
Quantifying effects in plots, including uncertainty
Link to cause-and-effect.
Visualization and values of effects.
Effects of multiple inputs.
Iterative model building as understanding increases
Prediction
Daily operation, decision making
Optimization, training AI
Robustness check (variations on inputs)
Prediction errors & its uncertainty

Figure 36 A taxonomy of data-based modelling. Line thickness at the bottom indicates suitability & typical use. Acronyms are as follows: DM (digital model), ML (machine learning), CNN (convolutional neural networks), GAN (generative adversarial network), VAE (variational auto encoder), R&D (research and development).

Figure 36 is related to the ideas in [67]. Some remarks on the taxonomy in Figure 36:

Surrogate models of detailed physics models or simulation models. Such models may well be available where design parameters as inputs can be identified as a selection of inputs. However, one evaluation may require considerable computational effort (e.g., minutes, hours). In this case, a carefully selected list (Design of Computer Experiments) can be constructed and fed to the detailed model. Then, the simulation model is run in batch mode (e.g., over the weekend), and the vector of output values observed and recorded, and then each output is modelled by a surrogate model that is quick to evaluate. For optimization of the model or reinforcement learning quick evaluations are necessary. Some further concepts

- Deterministic deviations or "errors" (i.e., the difference in outcome of the detailed and the surrogate model) are typical, i.e., if the detailed model is run again, exactly the same output occurs. This makes modelling an evaluation different from the observational (field) data setting as statistical uncertainty is not quantified.
- Model types include Kriging, a.k.a. Gaussian Process; GAM (Generalized Additive Model), neural networks. Not all models support categorical inputs and/or outputs. See [66].
- Understanding and quantifying effects can be done graphically although this may still be a challenge. Surrogate models can be very useful in understanding a black box detailed model; for instance, it may guide design decisions and iterations with further model insights.
- Multiple objective optimizations if each of the outputs is modelled and has some desirable direction, a balance can be found in an iterative way. This way of working goes beyond automatic optimization or the RL reward setting, where the objective / reward is given by a single number.
- Design of Computer Experiments. This is about constructing a limited set of "runs" for the detailed model that still allows to estimate a model of certain complexity of the mathematical relationship between input and output. For instance, the limited set of runs might include only a subset of all possible combinations of the 3-vector of (discrete) inputs (x1, x2, x3). An example of a model with low complexity is a linear expression in the inputs. Some techniques are space-filling LHD (space-filling Latin Hypercube Design; i.e., a set of points in a bounded multi-dimensional space with certain desirable properties) and quasi-Monte-Carlo methods. Further considerations are the ability to impose constraints, and to supplement existing datasets with new runs. See e.g. [10], [68], [69], and [70].

Domain knowledge / simple relations and data-oriented methods

- An often-useful distinction of the purpose of a model is *understanding* versus *prediction*. See [71], [72], and especially [73]. In understanding, the goal is to make decisions, gain understanding by recognizing the underlying mechanisms, perhaps improve the model. The other modelling purpose is prediction, where the practitioner might not care for the underlying mechanisms if the predicted values of future cases are close to true outcomes. Many machine learning techniques have been developed with large quantities of data and predictions in mind. In contrast, many statistical methods have been developed with a limited amount of data and the purpose of understanding in mind, and deal with the resulting uncertainty by reporting confidence intervals, and/or "statistical significance". The combination of the two viewpoints can be powerful when dealing with limited amounts of data. In Figure 36, the middle columns deal with these cases. In extreme, the prediction could be a black box as in some of the machine learning methods, although the explainable AI (see [74]) tries to alleviate this. However, a hybrid form where the model form is clearly understood and still gives good enough predictions can be preferred over a slightly better predicting black box (see [75]). In practice, the way that data is collected often results in groups, e.g., many values within one system or one situation. Another example is when a major calibration event occurs, all following observations until the next calibration share some influence of the calibration action and form one group. In the semiconductor industry a *lot* is a group of 25 wafers in one special container, which often share some properties. Often, the model errors of observations within a group are more similar than errors between groups because of a shared, unmodelled influence. In statistics, this is often explicitly modelled using random effects, and the increased uncertainty from the within-group similarities is taken into account. In machine learning texts, this grouping structure is mostly ignored. However, this should still be considered as it may severely impact generalization error (for new "groups") and also may increase overfitting. See [76] and [11].
- For prediction, classical statistical methods rely on correct model specification; in machine learning, the algorithms are designed to be robust for peculiarities in the data, often using hyper parameter optimization, cross validation, train/test sets. In situations where test sets are small, the prediction performance or generalization error can still have large statistical uncertainty. Often this statistical certainty is not considered. Suppose a prototype medical diagnostic method detects 90% of the cases correctly, but a decision needs to be made how promising further development is. It then matters for the decision making whether the 90% detection has a large

uncertainty of e.g. 40%-98%, or a small uncertainty, e.g. 88%-92%. The choice of methods for cross validation are not always obvious when data is scarce. See e.g., [77] and [11].

- Example methods for simple relations are regression, logistic regression, or CART, see e.g. [78]. Example methods for data-oriented models are neural nets, random forests, Lasso/Ridge, SVM (support vector machines).
- Empirical data comes in two main types. The first is *observational*, the second is by performing a designed *test*. The difference between these is whether the circumstances and inputs are manipulated by the experimenter (test), or otherwise the system is observed as events unfold themselves and observations are merely recorded. The latter is typical in many machine- learning applications.

Complex data as the fourth column in Figure 36 shows that there are more data-based model building approaches where inputs may be "complex" such as images. Here, "complex" is meant as more complex than the first three columns. Without giving a precise definition, in the non-complex cases above, the input of a single observation is a vector $(x_1, \ldots, x_p)$ of fixed length p of say less than 100, although the inputs are possibly heavily processed and may encode a mix of discrete and continuous variables. Typically, each of those variables is human-interpretable. In contrast, more complex examples are the free text as handled by the LLM (Large Language Models), images, audio, and video. For instance, an image of 100x100 pixels of grey values could be considered as a vector of length 10000 but is better dealt with by dedicated methods such as convolutional neural networks, also known under the term deep learning. A basic application is classification to identify objects on an image, and additional applications are a heatmap that explains which parts of the images lead to the classification. Another application as a generic building block is GAN (generative adversarial network). Given a modestly sized set of images of some sort, one model tries to make new images as similar as possible but has no access to the set of images. Another model, the adversary of the first model, tries to predict whether a given image is original or generated. Together these iteratively learn and improve, resulting in a model that can generate similar-like images that may be useful in AI training.

When modeling randomness one gets into the area of generative modelling. Generative modeling is the counterpart of discriminative modeling, which contains both regression and classification and is the more well-known part of machine learning. Generative modelling focusses on $p(x|y)$ rather than $p(y|x)$ as in discriminative modelling. Where x is the data and y is the label. For instance, in images of cats and dogs, where the images are the x and the labels (cat or dog) are the y, a generative model gives you an image of a cat or a dog given the label. In generative modeling, the label y is often not relevant, as the primary goal is to obtain the data distribution for a particular case. The simplest form of generative modeling involves estimating a distribution from data, while more complex models include Naive Bayes and Hidden Markov models. Recently, there has been an increase in the use of Deep Generative Models (DGM), such as Generative Adversarial Networks (GAN) [79] and Variational Autoencoders (VAE) [80] [81] . These models can handle very complex distributions, such as the distribution of 128x128-pixel images containing faces. One can use these models for instance to introduce variation. Suppose there is an input that can vary from time-to-time. One can use another instance of some dataset every time. However, there is little variation when the dataset is small. One can use a generative model to capture the underlying distribution and use a new sample as input every time. For example, a comparison of the use of different DGMs in a digital shadow can be found in [82].

4.4.3 Modelling the variation block

As shown in Figure 16, a block with *variation* on the DT is added during training, so that the RL agent encounters a wider variety of situations. Aspects of variation could be:

- Random changes to structural behavior of a DT within a system, e.g. a randomly chosen rate of change over time (ageing, degradation, environmental changes). A new random value would be chosen after a while. Instead of gradual changes, also sudden jumps can be simulated (e.g., replacing of a module with different properties)
- Random variation can also be applied between different systems, as a system-specific property. For instance, differences in hardware components between systems may mean that the effect of a control knob is slightly different between systems.

- This could be described using random slopes in an extended regression model, $output = (A + a_i) + (B + b_i) * control + error$ with ai a random intercept and bi a random slope, both on average 0, and following some probability distribution.
- The DT itself may also have randomness, e.g., the rate of ageing or degradation may be random in the UUV use case if gusts of winds or random positions of the sun or pedestrians are simulated. In logistics applications, demand is random. In the TEM use case, random fluctuations of electron paths / elements of the Ronchigram may be used. External variations may be interpreted as changes to the probability distributions underlying the DT simulation. The variations block could for example vary standard deviations for the DT randomness. This can become complicated, e.g., if the DT randomness needs a correlation matrix to simulate multiple random variables, the variation block should vary the correlation matrix itself. This is not straightforward as slight changes to individual entries of the correlation matrix may render the matrix invalid as a correlation matrix (technically, this matrix needs to be positive definite).
  - For example, in a DT describing a logistics case, the Poisson rates of demands of containers between two locations were varied, for every pair of from/to location. Here, the changes to Poisson rate were according to a chosen probability distribution themselves, adding an extra layer of randomness to the story. This was interpreted as a robustness check for the interplay with agents; if the nature of the random behaviour of the DT changes somewhat, will the agent's policy still give good outcomes? This is described in [83].

4.4.4 Modelling strategies for adaptation

In section 3.3.1.2 the notation c_{tune} is used to refer to configuration parameters of the DT as part of the set of numerical parameters of the DT that govern the relation between input and output. This is a situation in which a DT or a model is used in operation for predicting y as a function of inputs x. The relation between y and x may change over time. Some strategies of correcting the DT are as follows.

A few examples on how to change a model or DT over time to adjust to changing circumstances. For more details on model adaptation, also consider D2.1 [36].

1) Basic strategy: Rebuild the model periodically, with a window of most recent data of N periods. The window may be chosen so that the period resembles the upcoming period, because of machine settings, environmental changes, etc. Here the main structure of the model would stay the same, and using observations of input-output pairs, the model would be retrained (parameters estimated / calibrated / tuned) using recent data only.
2) A variant or hybrid of fixed and tunable parameters is found in a class of statistical methods with random effects. For instance, consider a simple input-output pair $y_{pred} = a_i + b_i x$. The sensitivity b_i might change between systems or situation i. This i may be a group as described in 4.4.2. When developing the DT, it may be established what plausible values are for b_i by using a probability distribution: each new system or situation i brings a new, random b_i that follows say a normal distribution with mean μ_b and standard deviation σ_b. Facing a new situation i with some examples (x, y), a basic strategy is to perform linear regression on the small dataset to find a_i and b_i. As an alternative, statistical methods use the earlier information μ_b, σ_b alongside the small dataset to estimate b_i (e.g., using Bayesian estimation or maximum likelihood). As a_i and b_i are linked, one should consider the pair (a_i, b_i) as 2-vector following a bivariate normal distribution needing 2 means, 2 standard deviations, and a correlation parameter; one may think of a scatterplot of possible (a_i, b_i) pairs. For this strategy, a few tens of groups i are usually enough, where fewer than 10 groups may well result in numerical problems. Remarks,
   a. In notation of section 3.3.1.2, this gives a hybrid solution beyond a clear separation of model parameters into a fixed and tunable part; b_i is tunable but the distributional information on b_i is added to the knowledge of the fixed part.

b. A common way of dealing with this situation is by taking the data of the latest and all previous situations i and using the statistical technique of “maximum likelihood estimation” to obtain values for all mentioned model parameters.
c. It is also possible to use statistical estimation where a tuning phase with situations i is more separate from the operational phase where a new situation i is observed, by use of Bayesian estimation. In it, the information of the tuning phase on the bivariate distribution (a_i, b_i) is estimated during the tuning phase and the distribution parameters are stored. This then serves as a Bayesian prior during the operational phase when analyzing observations from the latest situation i, and estimating the latest (a_i, b_i).
d. An example is described in [84] that considers software reliability: describing the errors found in software over time while testing. Specific to the case studied here, software is developed in an agile way, adding features every two weeks, so that errors can be linked to specific feature. These features constitute the groups. The total (unknown) number of errors in feature i is a model parameter a_i. There is also a b_i describing the rate at which errors are found during testing. Both parameters are considered random, and when a new feature still only has few errors found, data from the earlier groups help estimate the new features properties. The model output are quantities such as remaining number of errors in the software and expected number of errors to be found in 6 weeks of testing.

3) A small experiment (small in terms of number of input combinations, say a few tens) carried out on the system. Suppose you can identify 1-to-10-dimensional subspace of inputs whose effects may be different for different systems/situations i as in the previous point. It may be possible to carry out a small number of settings according to a clever scheme (“Design of Experiments”) to derive the system-specific effects; here each of the settings is given as input to the system and the output measured. In industry, an example may be mass manufacturing where an electronic device (e.g., the old CRT television) was measured using such an experiment. Effects were derived from the experiment and from here followed specific “settings” that were put in the embedded software with device-specific values. With few inputs (say <=4), all possible combinations of low & high settings may be formed. For more, this may become infeasible as it grows by 2 to the power of nr of inputs. Fortunately, the subfield of statistics called “design of experiments” gives many techniques to greatly reduce the number of required experiments, e.g., 32 in case of 10 inputs. One can imagine that sometimes a CPS also carries out a short array of experiments in the ASIMOV solution during a dedicated short phase for this recalibration.
4) Monitoring. In case a new system j is introduced one could use the developed DT for system i and monitor how well it works. After a certain time-period, this feedback can be used to determine the strategy for the development of the DT specific for system j; this could range from “DT_i works fine for j” to “DT_j requires a complete new fit of a_i and b_i”. Another direction for monitoring is as follows. Within the world of system i, one could monitor also when DT_i requires a new fit of model parameters, based on some quality metric (e.g., the reward function). For instance, the field of SPC (statistical process control) gives techniques for monitoring a noisy signal; these techniques have been applied in manufacturing settings for many decades (see e.g., the classic book [85]).

It may occur that smaller DT modules form the DT, but only data is observed on the final output (the DT as a whole). At the same time, some model constants/parameters of the DT modules may still be unknown and need to be estimated from data. This is an example where the forward problem is simple, given numeric values of the model parameters, outcomes are concrete). The backward problem is: given data, derive numeric estimates of the model parameters. In well-posed situations this may be solved using mathematical optimization. There is also the interesting approach of Bayesian estimation that may tackle several issues at once.

- For some numeric parameters one probably has knowledge of the range of plausible values (e.g., material properties). In Bayesian estimation, priors are formulated using a probability distribution that corresponds to these plausible ranges and this knowledge is then naturally incorporated.
- If there is a limited amount of data, the uncertainty of the numeric model parameters will be large. Bayesian estimation results in posterior distributions of these, that describe the uncertainty exactly based on prior knowledge and updated with the data.

- The numeric parameters may be difficult to separate, e.g., if for the first module y1=a1+b1*x and for the second module y2 = a2+b2*y1, then based on data consisting of pairs (x, y2) (so excluding the intermediate y1), the four parameters (a1, b1, a2, b2) cannot be uniquely determined. In less obvious cases, certain combinations may be difficult to determine. The posterior distribution is multivariate on all model parameters and makes it easy to describe this problem. In this example, the posterior distribution describes which subset of values (a1, b1, a2, b2) are plausible given the data.
- One example is a Nature publication [86] in the beginning of the Covid pandemic. They collected data of multiple countries on different non-pharmaceutical interventions (lockdowns, closing schools, shops, offices) which the countries did in different orders and time frames. These timings were the predictors. The outcome was covid mortalities on national level, so that only the total effect of the country's interventions was observed. Using these Bayesian methods, they tried to disentangle the various effects.

4.5 Challenging behavior

In the very general context of constructing digital twins (shadows, models), the challenging behavior might follow from a detailed description from first principles. A bottom-up approach from DT from modules may be helpful here. In addition, the presence of chaos or disturbances call for stochastic elements of the DT: randomness added to results according to carefully specified probability distributions (e.g., raindrops, gusts of wind; randomness in the EM use case, random demand in logistics).

In case of smaller DT modules or sub-DTs and when data is available to build or calibrate the model: the problem may be simple enough that the predictor form is not too complex (e.g., vector of dimension of max few 10s) and that in principle the relation is known. Then some of the above challenges can be addressed using "tricks" from statistics and machine learning.

- **Disturbing events**, if with a small chance a rare event occurs so that the outcome is very different, the model may be a composite one; a component describing whether the rare event occurred and a component with a usual outcome. If the event is not too rare and present in the data, statistical methods may be possible to use. For example, suppose in a US holiday destination, one is interested in the number of fish caught in a day by each visitor is of interest. The number of fish may be modelled using a Poisson distribution, with predictors on the details of the fishing rods, location, weather, depth of the water, etc. The unexpected event could be that the person did not even start to fish, which may be predicted by a different set of predictors (e.g. type of visitor, activities of interest, bad weather, distance to suitable fishing location). This non-fishing event necessarily results in a zero count of caught fish. A ZIP (Zero-inflated Poisson) model describes this situation, where one model component describes the probability of the non-fishing event as function of predictors, and another the number of fish caught given that the visitor tried to catch one, also related to a set of predictors. The ZIP model is an example of a model consisting of smaller model blocks where different random "regimes" are disentangled and made explicit if data is available.
- **Delayed response**. In the case where predictors and response are time series with unknown delays, this can in principle be investigated by taking a predictor and add "lagged copies" of it: a copy of the predictor as it was k timesteps ago, for a properly chosen set of k. If the set of lags is large, the predictor set gets large and a selection method may be appropriate (e.g., Lasso regression).
- **Stability**. If the CPS is not disturbed by inputs or control actions, and left to run, relevant outputs may show stable behavior, or drift, or jumps. If this occurs, it is difficult to predict future values, even in the case when no control actions are taken.
- **Hysteresis.** This is the phenomenon that the output depends on not just the most recent input, but also inputs of previous points in time: the whole history. An example is how magnetic lenses react on applied electric currents and magnetic fields. The material's magnetic properties react to changing currents, but in a delayed way with left overs from earlier situations.

5 Conclusion

This deliverable presents the ASIMOV work regarding a DT reference architecture for DT-based AI-training for CPS optimization. The report describes the challenges in creating a model mimicking the behavior of a real system. In the state-of-the-art section only a few, high-level architectural descriptions are described, due to the novelty of this field. In four viewpoints we highlight the necessary considerations and requirements for the system, a model view on functionality, a logical design solution, and technical considerations for the implementation. This architectural part is followed by an extensive elaboration on the actual considerations of mimicking system behavior. The various types of behavioral modeling are described in detail.

To conclude, this report is an extensive source of information, considerations, and guidelines for system architects, system engineers and other developers responsible for the development of digital twins or digital models intended for training AI systems.

6 Terms, Abbreviations and Definitions

Table 1 - Terms, Abbreviations and Definitions

CPS	**Cyber Physical System, a type of system**
DT	**Digital Twin**
PT	**Physical Twin. Is a ‘role’ in digital twinning.**
RLA	**Reinforcement Learning Agent**
VP	**Viewpoint**

7 Bibliography

[1] C. Hou, H. Remöy and H. Wu, “Digital twins to enable smart built heritage management: a systematic literature review,” in *Proceedings of the 27th Pacific Rim Real Estate Society Annual Conference*, Delft, the Netherlands, 2021.

[2] A. Bolton, M. Enzer and J. Schooling, “The Gemini Principles,” 2018.

[3] S. Aheleroff, X. Xu, R. Y. Zhong and Y. Lu, “Digital Twin as a Service (DTaaS) in Industry 4.0: An Architecture Reference Model,” *Advanced Engineering Informatics,* vol. 47, 2021.

[4] Erikstad, Stein Ove, “Merging Physics, Big Data Analytics and Simulation for the Next-Generation Digital Twins,” in *HIPER 2017, High-Performance Marine Vehicles*, Zevenwacht, South-Africa, 2017.

[5] R. Doornbos and et.al., “ASIMOV Reference Architecture,” ASIMOV consortium, 2022.

[6] M. W. Maier and E. Rechtin, The art of systems architecting, CRC Press, 2000.

[7] J. Pearl and D. Mackenzie, The Book of Why: The New Science of Cause and Effect, New York: Basic Books, 2018.

[8] J. Pearl, M. Glymour and N. P. Jewell, Causal Inference in Statistics: A Primer, John Wiley & Sons, 2016.

[9] L. Wilkinson, The Grammar of Graphics, New York: Springer New York, 2005.

[10] E. D. Stinstra, L. T. Driessen and P. H. Stehouwer, “Design optimization: some pitfalls and their remedies,” in *Proceedings of the 3rd ASMO UK/ISSMO conference*, Harrogate, North Yorkshire, UK, 2001.

[11] J.-W. Bikker, B. Schriever and M. Tijink, “Empirical models - understanding and validation,” CQM, www.cqm.nl/asimov/Empirical_models_understanding_and_validation, Eindhoven, 2023.

[12] K. Pohl, M. Broy, H. Daembkes and H. Hönninger, Advanced Model-Based Engineering of Embedded Systems, Springer International Publishing, 2016.

[13] E. Ferko, A. Bucaioni and M. Behnam, “Architecting Digital Twins,” *IEEE Access,* vol. 10, p. 50335–50350, 2022.

[14] Z. Lei, H. Zhou and W. Hu, “Toward a web-based digital twin thermal power plant,” *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS,* Vols. 18(3):1716 - 1725, 2022.

[15] F. Laamarti and et.al., “An iso/ieee 11073 standardized digital twin framework for health and well-being in smart cities,” *IEEE Access,* 2020.

[16] Y. Pan and et.al., “Digital twin based real-time production logistics synchronization system in a multi-level computing architecture,” *Journal of Manufacturing Systems,* vol. 58, 2020.

[17] H. Zhang and et.al., “Dynamic resource allocation optimization for Digital Twin-driven smart shopfloor,” in *IEEE International Conference on Networking, Sensing and Control (ICNSC)*, 2018.

[18] B. Song, F. Tao and F. Sui, “Digital twin-driven product design framework,” *Artic. Int. J. Prod. Res.,* 2018.

[19] A. Al-Ali, R. Gupta and T. Zaman Batool, “Digital Twin Conceptual Model within the Context of Internet of Things,” *Future Internet 12:163,* 2020.

[20] B. Ashtari Talkhestani and T. L. B. Jung, “An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System,” *Automatisierungstechnik 67:762–782,* 2019.

[21] G. Steindl, M. Stagl and L. Kasper, “Generic Digital Twin Architecture for Industrial Energy Systems,” *Applied Sciences 10:8903,* 2020.

[22] S. Abburu, A. Berre and M. Jacoby, “Hybrid and Cognitive Digital Twins for the Process Industry,” in *IEEE International Conference on Engineering, Technology and Innovation*, 2020.

[23] B. Osiński, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu and H. Michalewski, “Simulation-based reinforcement learning for real-world autonomous driving,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[24] K. Alexopoulos, N. Nikolakis and G. Chryssolouris, “Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing,” *Int. J. Comput. Integr. Manuf.,* vol. 33, no. 5, p. 429–439, 2020.

[25] H. Ju, R. Juan, R. Gomez, K. Nakamura and G. Li, “Transferring policy of deepreinforcement learning from simulation to reality for robotics,” *Nature MachineIntelligence,* vol. 4, no. 12, p. 1077–1087, 2022.

[26] X. Gan, Y. Zuo, A. Zhang, S. Li and F. Tao, “Digital twin-enabled adaptive scheduling strategy based on deep reinforcement learning,” *Science China Technological Sciences,* p. 1–15, 2023.

[27] G. Shen, L. Lei, Z. Li, S. Cai, L. Zhang, P. Cao and X. Liu, “Deep reinforcement learning for flocking motion of multi-uav systems: Learn from a digital twin,” *IEEE Internet of Things Journal,* vol. 9, no. 13, p. 11141–11153, 2021.

[28] G. Shen, L. Lei, X. Zhang, Z. Li, S. Cai and L. Zhang, “Multi-uav cooperative searchbased on reinforcement learning with a digital twin driven training framework,” *IEEE Transactions on Vehicular Technology,* 2023.

[29] R. Julian, B. Swanson, G. Sukhatme, S. Levine, C. Finn and K. Hausman, “Neverstop learning: The effectiveness of fine-tuning in robotic reinforcement learning,” *arXiv:2004.10190,* 2020.

[30] S. R. Jamil and M. Fawad, “A comprehensive survey of digital twins and federated learning for industrial internet of things (iiot), internet of vehicles (iov) andinternet of drones (iod),” *Applied System Innovation,* vol. 5, no. 3, p. 56, 2022.

[31] E. Ferko, A. Bucaioni and M. Behnam, “Architecting digital twins,” *IEEE Access,* vol. 10, p. 50335–50350, 2022.

[32] Z. Wang, S. Chang, Y. Yang, D. Liu and T. S. Huang, “Studying Very Low Resolution Recognition Using Deep Networks,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4792-4800*, 2016.

[33] M. E. Taylor and P. Stone, “Transfer learning for reinforcement learning domains: A survey,” *Journal of Machine Learning Research,* vol. 10, no. 7, 2009.

[34] W. Zhao, J. P. Queralta and T. Westerlund, “Sim-to-real transfer in deep reinforcement learning for robotics: a survey,” in *IEEE symposium series on computational intelligence (SSCI) (pp. 737-744)*, 2020.

[35] I. A. Thijs and et.al., “Requirements for AI-technology for DT-based AI-training and for system optimisation/configuration,” 2022.

[36] S. Moritz, “Identification of relevant parameters modelled in DT,” ASIMOV project, 2022.

[37] H. Vanrompay and et.al., “Specifications and Commonality Analysis,” ASIMOV project, 2022.

[38] IEEE-SA Standards Board The Institute of Electrical and Electronics Engineers, “IEEE Standard for Functional Modeling Language—Syntax and Semantics forIDEF0, Software Engineering Standards Committee of the IEEE Computer Society,” 25 6 1998. [Online]. [Accessed 4 5 2022].

[39] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, “CARLA: An open urban driving simulator,” in *Conference on robot learning*, 2017.

[40] N. Braun, “Application to Systems - Methods and Techniques”.

[41] F. Wilking, C. Sauer, B. Schleich and S. Wartzack, “Sysml 4 Digital Twins – Utilization of System Models for the Design and Operation of Digital Twins,” in *Proc. Des. Soc. 2, pp. 1815–1824*, 2022.

[42] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Boston, MA: Prentice Hall, 2017.

[43] “MQTT: The Standard for IoT Messaging,” [Online]. Available: https://mqtt.org/.

[44] J. L. Ortega-Arjona, Patterns for Parallel Software Design (1st. ed.), Wiley Publishing, 2010.

[45] J. K. E. Sanders, CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 2010.

[46] Microsoft, “Learn about twin models and how to define them in Azure Digital Twins,” Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/azure/digital-twins/concepts-models.

[47] S. V. Nath and P. v. Schalkwyk, Building Industrial Digital Twins: Design, develop, and deploy digital twin solutions for real-world industries using Azure Digital Twins, Packt Publishing Limited.

[48] “Digital Twin Consortium,” [Online]. Available: https://www.digitaltwinconsortium.org/. [Accessed 28 3 2024].

[49] F. Buschmann, K. Henney and D. C. Schmidt, Pattern-Oriented Software Architecture, Volume 4: A Pattern Language for Distributed Computing, Wiley, 2007.

[50] ISO, “ISO/DIS 23247-1,” [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:23247:-1:dis:ed-1:v1:en.

[51] ISO/IEC, “ISO/IEC JTC 1/SC 41,” [Online]. Available: https://www.iec.ch/dyn/www/f?p=103:7:413461466279946::::FSP_ORG_ID,FSP_LANG_ID:20486,25.

[52] “Eclipse BaSyx,” [Online]. Available: https://github.com/eclipse-basyx. [Accessed 28 3 2024].

[53] “BaSyx Architecture,” [Online]. Available: https://eclipse.dev/basyx/architecture/.

[54] “BaSyx / Overview,” [Online]. Available: https://wiki.eclipse.org/BaSyx_/_Overview.

[55] S. Moritz, “ASIMOV: deliverable 4.2”.

[56] P. v. Schalkwyk, “Digital Twin Capabilities Periodic Table, A Digital Twin Consortium User Guide”.

[57] R. Ackoff, “Towards a System of Systems Concepts,” *Management Science,* vol. 17, no. 11, 1971.

[58] A. Goel, S. Rugaber and S. Vattam, “Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language,” *AI EDAM,* vol. 23, no. 1, pp. 23-35 , 2009.

[59] J. S. Osmundson, T. V. Huynh and G. O. Langford, “emergent behavior in systems of systems,” in *INCOSE International Symposium*.

[60] M. Segovia and J. Garcia-Alfaro, “Design, Modeling and Implementation of Digital Twins,” *Sensors,* no. 22, p. 5396, 2022.

[61] M. Eckhart and A. Ekelhart, “A Specification-based State Replication Approach for Digital Twins,” in *Workshop on Cyber-Physical Systems Security and PrivaCy*, 2018.

[62] S. Chakraborty and S. Adhikari, “Machine learning based digital twin for dynamical systems with multiple time-scales,” *Computers & Structures,* vol. 243, 2021.

[63] N. Khakpour, S. Jalili, C. L. Talcott, M. Sirjani and M. Mousavi, “Pobsam: Policy-based managing of actors in self-adaptive systems,” *Comput. Sci., 263:129–143, 2010.*

[64] N. Khakpour, J. Kleijn and M. Sirjani, “A Formal Model to Integrate Behavioral and Structural Adaptations in Self-adaptive Systems,” in *Fundamentals of Software Engineering*, 2019.

[65] B. Kuipers, “Qualitative simulation,” *Artificial Intelligence,* vol. 29, no. 3, pp. 289-338, 1986.

[66] Á. Bárkányi, T. Chován and S. Németh, “Modelling for Digital Twins—Potential Role of Surrogate Models,” *Processes,* vol. 9, no. 3, p. 476, 3 2021.

[67] J. de Mast, S. H. Steiner, W. P. Nuijten and D. Kapitan, “Analytical Problem Solving Based on Causal, Correlational and Deductive Models,” *The American Statistician,* vol. 77, no. 1, pp. 51-61, 2023.

[68] D. Den Hertog and A. Y. Siem, “Kriging models that are robust with respect to simulation errors,” *CentER discussion paper,* Vols. 2007-68, 2007.

[69] J. Van Sambeeck, J.-W. Bikker and P. Stehouwer, “Status and challenges in surrogate modelling,” CQM, www.cqm.nl/asimov/Status_and_challenges_in_surrogate_modelling, Eindhoven, 2023.

[70] R. B. Gramacy, Surrogates: Gaussian process modeling, design, and optimization for the applied sciences, CRC press, 2020.

[71] L. Breiman, “Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author),” *Statistical Science,* vol. 16, no. 3, pp. 199-231, 2001/08.

[72] G. Shmueli, “To Explain or to Predict?,” *Statistical Science,* vol. 25, no. 3, 2010.

[73] B. Efron, “Prediction, Estimation, and Attribution,” *Journal of the American Statistical Association,* vol. 115, no. 530, pp. 636-655, 2020.

[74] C. Molnar, Interpretable Machine Learning: A Guide For Making Black Box Models Explainable, Munich, Germany: Independently published, 2022.

[75] R. Couronné, P. Probst and A.-L. Boulesteix, “Random forest versus logistic regression: a large-scale benchmark experiment,” *BMC Bioinformatics,* vol. 19, no. 1, p. 270, 2018.

[76] T. A. Rutten, “Mixed-effects random forest model for quantifying relations in clustered data,” Eindhoven university of technology, Eindhoven, 2021.

[77] L. Lusa, L. M. McShane, M. D. Radmacher, J. H. Shih, G. W. Wright and R. Simon, “Appropriateness of some resampling-based inference procedures for assessing performance of prognostic classifiers derived from microarray data,” *Statistics in Medicine,* vol. 26, no. 5, pp. 1102-1113, 2007.

[78] T. Hastie, R. Tibshirani and J. Friedman, Elements of Statistical Learning: data mining, inference, and prediction. 2nd Edition., Springer, 2009.

[79] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative Adverserial Nets,” *Proceedings of the International conference on Neural Information Processing Systems (NIPS),* pp. 2672-2680, 2014.

[80] L. Pinheiro Cinelli and e. al., “Variational Autoencoder,” in *Variational Methods for Machine Learning with Applications to Deep Networks*, Springer, 2021, p. 111–149.

[81] D. P. Kingma, D. J. Rezende, S. Mohamed and M. Welling, “Semi-Supervised Learning with Deep Generative Models,” *Advances in neural information processing systems,* vol. 27, 2014.

[82] R. Van Beek, “Generative Modelling for Digital Twins, a use-case,” CQM, www.cqm.nl/asimov/Generative_Modelling_for_DT_a_use_case, EIndhoven, 2023.

[83] F. De Ruiter, “Practical reinforcement learning with large action space,” CQM, www.cqm.nl/asimov/Practical_reinforcement_learning_with_large_action_space, Eindhoven, 2023.

[84] W. van Driel, J. W. Bikker, M. Tijink and A. Di Bucchianico, “Software Reliability for Agile Testing,” *Mathematics,* vol. 8, no. 5, 2020.

[85] D. C. Montgomery, Introduction to statistical quality control, Eighth edition ed., Hoboken, NJ: John Wiley & Sons, Inc., 2020.

[86] S. Flaxman, S. Mishra and A. e. a. Gandy, “Estimating the effects of non-pharmaceutical interventions on COVID-19 in Europe,” *Nature584,* no. https://doi.org/10.1038/s41586-020-2405-7, p. 257–261, 2020.

[87] ASIMOV-consortium, *ASIMOV - Full Project Proposal,* 2020.

[88] D. Jones, C. Snider, A. Nassehi, J. Yon and B. Hicks, “Characterising the Digital Twin: A systematic literature review,” *CIRP Journal of Manufacturing Science and Technology,* vol. 29, no. DOI: 10.1016/j.cirpj.2020.02.002., p. 36–52.

[89] L. Wright and S. Davidson, “How to tell the difference between a model and a digital twin,” *Adv. Model. Simul. Eng. Sci. ,* vol. 7 (1), no. DOI: 10.1186/s40323-020-00147-4.

[90] B. R. Barricelli, E. Casiraghi and D. Fogli, “A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications,” *IEEE Access,* vol. 7, no. DOI: 10.1109/ACCESS.2019.2953499, p. 167653–167671, 2019.

[91] M. Grieves and J. Vickers, “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In Franz Josef Kahlen, Shannon Flumerfelt, Anabela Alves (Eds.): Transdisciplinary Perspectives on Complex Systems,” in *Transdisciplinary Perspectives on Complex Systems*, 2016, pp. 85-113.

[92] D. J. Wagg, K. Worden, R. J. Barthorpe and P. Gardner, “Digital Twins: State-of-the-Art and Future Directions for Modeling and Simulation in Engineering Dynamics Applications,” *ASCE - ASME*

Journal of Risk and Uncertainty in Engineering Systems, vol. 6(3), no. DOI: 10.1115/1.4046739, 2020.

[93] K. Pohl, H. Broy, Daembkes and H. Hönninger, Advanced Model-Based Engineering of Embedded Systems, Springer International Publishing, 2016.