A Generic DigitalTwin Model for Artificial Intelligence Applications

1st Oliver Niggemann
Institute of Automation Technology,
Helmut Schmidt University, Hamburg, Germany
oliver.niggemann@hsu-hh.de

2nd Alexander Diedrich
Fraunhofer IOSB, Fraunhofer Center for Machine Learning, Lemgo, Germany
alexander.diedrich@iosb-ina.fraunhofer.de

3rd Christian Kühnert
Fraunhofer IOSB, Fraunhofer Center for Machine Learning, Karlsruhe, Germany
christian.kuehnert@iosb-ina.fraunhofer.de

4th Erik Pfannstiel
Institute of Automation Technology,
Helmut Schmidt University, Hamburg, Germany
erik.pfannstiel@hsu-hh.de

5th Joshua Schraven
Institute of Automation Technology,
Helmut Schmidt University, Hamburg, Germany
joshua.schraven@hsu-hh.de

Abstract—Services for Cyber-Physical Systems based on Artificial Intelligence and Machine Learning require a virtual representation of the physical. To reduce modeling efforts and to synchronize results, for each system, a common and unique virtual representation used by all services during the whole system life-cycle is needed—i.e. a DigitalTwin. In this paper such a DigitalTwin, namely the AI reference model AIITwin, is defined. This reference model is verified by using a running example from process industry and by analyzing the work done in recent projects.

I. INTRODUCTION

DigitalTwins are a key concept for Cyber-Physical-Systems (CPS): By maintaining a collection of relevant information about a physical entity, a digital shadow is created which can be used for tasks such as monitoring, diagnosis or optimization. Most publications about the DigitalTwin focus on engineering- and process-oriented aspects such as a continuous enrichment of the twin during its life-cycle [30], on simulation scenarios [28], [20] or on modeling questions such as optimal meta-levels [29], hierarchies [30] or engineering chains [20]. Even the few publications written with a clear focus on Artificial Intelligence (AI) and Machine Learning (ML) fail to connect the content and capabilities of the DigitalTwin with AI/ML methods. A review of the related work confirming this impression is given in section II. Under the bottom-line, the DigitalTwin is seen as an information sink into which all available information is poured—hoping for a benefit by AI/ML applications at some later point in the life-cycle.

On the other hand, AI/ML methods have always used models of the environment and domain knowledge. So the DigitalTwin concept and AI/ML should be compatible. But two main contradictions remain: First, AI/ML are very heterogeneous, and each AI/ML method comes with a specialized model formalism to capture relevant aspects of the environment and the application domain. Hence, the question is how a DigitalTwin can provide the correct model to each AI/ML method. The second contradiction is that AI/ML require explicit, i.e. by an algorithm processable, knowledge since compiled knowledge in form of simulation libraries, raw data or executables does not help. But most publications refer to these kind of information.

In this paper we follow the key idea in Edward E. Lee’s seminal paper [16] in which he claims that “The intellectual heart of CPS is in studying the joint dynamics of physical processes, software, and networks.”, i.e. from an AI/ML perspective the main task of a DigitalTwin is to use models to predict the behavior of the corresponding CPS subsystem, comprising cyber and physical parts. Here we add an additional constraint: The prediction knowledge must be explicit and processable by an AI/ML algorithm.

So the paper addresses the following research questions (RQs):

**RQ 1:** Can we develop, as shown in figure 1, a common API of a DigitalTwin which is suitable for a set of heterogeneous AI algorithms? I.e. here in this paper neither the AI algorithms nor the internal structure of the DigitalTwin is addressed but only a common DigitalTwin API.

**RQ 2:** Can AI algorithms use a common information base, stored in the DigitalTwin? Can AI algorithms even exchange information that way?

**RQ 3:** Can such a DigitalTwin be used for both machine learning algorithms (i.e. sub-symbolic AI) and symbolic AI algorithms? Bridging sub-symbolic and symbolic AI is currently a major topic in the AI community [15], [3].

In section III, an analysis of the requirements of AI/ML methods to the corresponding models is made and in section
IV the main solution approach is presented. As a result, a DigitalTwin AI reference model, called the AITwin, will be derived. If such a reference model is used by several AI/ML methods, multiple advantages arise: First of all, the results of one method could be used by another. Second, redundant information used by different methods are avoided, easing the modeling efforts and preventing contradicting behavior. Case studies can be found in section V, they are used to evaluate the solution approach. A conclusion is given in section VI.

II. RELATED WORK

As described in the introduction, the DigitalTwin with its enabling technologies plays a decisive role for the success in several areas and is an ongoing field of research. One of the most comprehensive reviews for the current-state-of-the-art has been made by Fuller [13] covering the areas manufacturing, smart cities and healthcare. As one of his conclusions he states that there is still the lack of DigitalTwin reference models, which leads to discrepancies between similar projects and hence slows down the progress of this technology. Proposals for DigitalTwin reference models have already been made in [4] with the focus on risk control and prevention on process plants, in [18] for smart-manufacturing and in [2] for cloud-based CPS.

Recent publications, taking into account DigitalTwins and AI/ML are given by Min [22] focussing on production optimization in petrochemical industry or [10] who proposes a machine-learning enhanced DigitalTwin for Human-Robot-Collaboration. Still, most of the research is dealing with DigitalTwins from a modeling perspective. For example [19] develops the DigitalTwin of a CNC-machine, in [34] a aircraft has been reengineered and [26] explains the challenges and enablers for DigitalTwin in modeling.

Hence, current research combining AI with DigitalTwins results always in individual solutions for specific application fields. The herein presented work tries to close this gap by proposing the DigitalTwin AI reference model AITwin.

III. AI/ML METHODS AND THE DIGITALTWIN

Here we categorize AI/ML methods and derive requirements to the DigitalTwin. The number of used AI/ML methods are legion and so are the formalisms used to model domain knowledge and environment models. Still for simplicity sake, we focus on those methods mainly used for CPS and categorize these AI/ML methods into three main categories shown on the left hand side of figure 2. To illustrate these applications, a running example is used in this paper (figure 3):

**Distance-to-Normal Machine Learning**: Anomaly detection is a typical example for this approach: The machine learner memorizes historical data points and computes a distance measure between a current data point (e.g. production operation point) and memorized points. Often, the historical data points are interpreted as “normal” and too distant data points are classified as “anomalous”. To some small extent, interpolation is used to deal with data points close to previous data points. To implement “Distance-to-Normal Machine Learning”, the Digital Twin (see right hand side of figure 2) must comprise a common timing model to synchronize all data.

In our running example, the vector of all flows can be used to detect unusual flows, i.e. anomalies.

**Extrapolation Machine Learning**: Using a DigitalTwin in autonomous systems for optimization and self-configuration
means using it in a closed control loop, e.g. for model-
predictive control tasks. I.e. the DigitalTwin must predict
output values such as resource consumption, positions, etc.
for new input values. Please notice that this extrapolation ca-
pability means to predict values, in time or distance, far away
from observed data points. In terms of ML this is especially a
problem for technical systems where often only few individual
data points are available which are also clustered in only a few
regions.

In our running example, the vector of all flows can be
predicted (i.e. extrapolated) for new valve positions. This can
be used to optimize the system configuration.

Symbolic Artificial Intelligence: Many AI tasks such as diag-
nosis, reconfiguration or planning require causal information,
i.e. symbolic AI models: Diagnosis uses root cause-symptom
relations ("if the tube leaks, the tank fills slowly") and planning
needs information about process steps ("Modul 1 takes a
board and drills a hole"). Such causalities always describe
effects resulting from causes where both effects and causes
are symbolic predicates in some formal logic. So the Digital
Twin (see right hand side of figure 2) must comprise such
logic-based models.

In our running example, the effect of switching a valve such
as an overflowing container can be modeled, e.g. to identify
root causes of errors.

IV. AI REFERENCE MODEL

In this section, the three research questions from section I
are discussed and, as a result, the DigitalTwin AI reference
model, AITwin, is derived.

A. Timing Model

While time is the essential feature of any physical pro-
cess (the "P" in CPS), software (the "C" in CPS) does not
come with any build-in concept of time. In software parallel
events may occur at various points in time and time spans
between commands are not defined but vary according to
process behavior. So first, a common time model of the
DigitalTwin must be established. Following an idea from [16],
one may differentiate between two general solutions for this:

Cyberizing the Physical: Normally such approaches define
signal values \( x \) as \( x(t, k) \) where \( t \in \mathbb{R} \) is the physical
time and \( k \in \mathbb{N} \) the number of an ordered sequence of
events/instructions in the software. This is called a superdense
model of time.

Physicalizing the Cyber: New means for the time synchroniza-
tion in distributed systems, e.g. the Precision Time Protocol
[14], allow for a consistent concept of physical time especially
in controllers. Hence, software now may refer to precise points
in time and time spans.

"Cyberizing the Physical" has the drawback that most AI
algorithms need to be adapted, since they require a clear
time model and concept of causality. Therefore, the usage of
"Physicalizing the Cyber" approach should be preferred, i.e. all
information \( x(t) \) in the DigitalTwin refers to a unique point
in time \( t \). This requires from the underlying technical systems
time synchronization between all devices and an implemented
concept of simultaneity.

So on this level only two interfaces are needed for the Dig-
italTwin AI reference model (AITwin): Function \( \text{getData}(i,t) \)
returns the \( i \)’th signal \( x \in \mathbb{R} \) at a specific point in time \( t \)
and function \( \text{getData}(t) \) returns all signals \( x \in \mathcal{X} \) at a specific
point in time \( t \). We refer with \( \mathcal{X} \) to the set of all possible
values \( x(t) \).

Please note that the vector \( x(t) \) captures the complete
observable state of the system, e.g. comprising sensor/actuator
signals, observable product/object locations and states, config-
uration parameters and so on.

B. Extrapolatable Models

While superficially much data seems to be generated by
Cyber Physical Systems, a different picture can be seen when
ML is applied: Data intensive ML such as deep learning
often fails or delivers unusable results in production settings
while in other domains it shows break-thru performance. This
is mainly due to the following two reasons: (1) Most data
comprises repetitive patterns –mirroring the repetitive structure
of typical production processes, while the pure amount of data
is sufficient. (2) The data hardly comprises information about
failures or other problems.

Therefore, learned models often only have limited extrap-
olation capabilities—which is essential for CPS. Furthermore,
even if learned models such as neural networks predict output
values for new input values, the prediction quality quickly
decreases for values far away from observed data points in
time and distance. Hence, the AI reference model, AITwin,
must not only be able to extrapolate but also to assess the
quality of the extrapolation. So far the best approach is the
usage of probabilistic models, e.g. probabilistic or Bayesian
Neural Networks [23] which can also be applied to time
series analysis [12]. So all predictions should not only be
implemented as a point estimator but should also compute the
probability that the prediction is correct. In general, we have
to differentiate between two types of analyses and predictions:

Static Predictions: For tasks such as condition-monitoring or
anomaly detection, only the signal values \( x(t) \in \mathbb{R}^n \) at some
point in time \( t \) are used. In other words, for the analysis a
static feature vector \( [11] \) is taken into account. Thereby, the
assumption is that no information is coded in the sequence of
value and all necessary information is contained in the current
signal values. This assumption is true for many CPSs, even
for systems which have a dynamic nature them-self. Typically
three use cases are implemented here by machine learning applications:

Anomaly Detection: For a new data point \( x \in \mathbb{R}^n \) its probability \( p(x|X) \) given some historical data \( X \) is (at least approximately) computed.

Prediction: Given a partial feature vector \( x' \in \mathbb{R}^n \), i.e. a feature vector with missing entries, those entries are computed in form of a complete feature vector \( x \), best with their probability (e.g. to handle ambiguous predictions). The given entries then become inputs and the computed entries become outputs. Of course, if some input to output configuration is not possible, this must be signaled to the user.

Optimization: The reader may note that the partial gradient over the signals can be created by extrapolating in the vicinity of an operation point, so no special functionality of the DigitalTwin is needed here for the optimization tasks.

Dynamic Predictions: A totally different situation arises when important information is coded in the sequence of signal values over time. Again, three use cases exist:

Anomaly Detection: Given historical data \( X = \{ x(t = t_0), x(t = t_1), \ldots, x(t = t_k) \} \), for a time step \( \Delta t \) and a new data vector \( x(t = t_k + \Delta t) \) and its probability is computed so that \( x(t = t_k + \Delta t) \) is the logical continuation of the series.

Prediction: Given historical data \( X = \{ x(t = t_0), x(t = t_1), \ldots, x(t = t_k) \} \) and a time step \( \Delta t \) the next value \( x(t = t_k + \Delta t) \) is computed, best with its probability distribution.

Optimization: The reader may note that the temporal gradient over the signals can be created by extrapolating in the vicinity of the point in time \( \Delta t \), so no special functionality of the DigitalTwin is needed here.

As mentioned earlier, for reliable predictions, the model must not only capture the behavior in the normal but also in the failure modes. For example predictions for a failure mode, like a broken drive or a blocked pipe, are helpful for diagnosis tasks. Hence, the DigitalTwin must be able to activate specific failure modes by setting some components to failure states.

Figure 6 shows the resulting API methods with extrapolation capabilities. The static case is solved by a function \( \text{extrapolateStatic}(x') \) where \( x' \) is a partially filled signal vector. The function returns the missing values \( x \) and a corresponding vector of probabilities, both with null entries if not computable. This solves all three use cases discussed above.

For the dynamic case, a function \( \text{extrapolateDynamic}(X, \Delta t) \) where \( X \) is a sequence of historical signal vectors up to the point in time \( t \) is defined. The function returns the next signal vector \( x(t + \Delta t) \) and a corresponding vector of probabilities, both with null entries if not computable. Again, this solves the three use case discussed above.

The failure modes are set by a function \( \text{setFailedComps}(C') \) where \( C' \) is a subset of all components. If a failure mode is set, the functions \( \text{extrapolateStatic} \) and \( \text{extrapolateDynamic} \) predict the behavior assuming that the components are not working correctly. Another function \( \text{getComps()} \) returns all components.

C. Causalities

No definition of causality exists which is accepted by the majority of researchers. We therefore first analyze the special usage of causality models in two main AI application fields of DigitalTwin and derive from there a suitable causality definition.

AI Application 1 - Consistency-based Diagnosis: Diagnosis is the task of computing the error cause based on incomplete symptoms, i.e. observations. Therefore, knowledge about the causalities between root causes (i.e. failures) and symptoms must be known since failures such as sensor faults or machine degradation often lead to complex symptom patterns, e.g. alarms, error messages and plant stops. Currently users often see complex alarms and symptom patterns but fail to identify quickly the correct root cause and therefore cannot repair the factory efficiently.

Often models predict the behavior only for the normal mode. Consistency-based diagnosis [8] is able to use so-called weak fault models, i.e. models which capture the system behavior only for the OK-mode. In most cases, it uses partial system models such as \( OK(C_{i_1}) \land \ldots \land OK(C_{i_k}) \rightarrow (s_1 \rightarrow s_2 \land \ldots \land s_l) \), \( k,l \in \mathbb{N} \) where \( OK(C) \) denotes a correct functioning of component \( C \) and \( s_i \in \{0,1\} \) is a binary/symbolic representation of the current system status, e.g. \( s_i \equiv \text{"Sensor Value 42 > 10"} \) or \( s_j \equiv \text{"Motor 3 is on"} \).

So normally \( s_i \in S \) where \( S \) describes all system states. In other words, if components \( C_{i_1} \ldots C_{i_k} \) work correctly, the implication describes the normal system causality. Consistency-based Diagnosis then redraws OK-assumptions until no contradictions between predictions and observation exist anymore.

AI Application 2 - Planning: Planning is the task of computing a sequence of process steps which transforms given raw objects into an also given final object. Let \( P \) be the set of all possible objects. Then a process step \( q_i \) is an implication \( q_i : x \rightarrow y \), which takes \( u \) input objects \( x \in P^u \) and transforms them into \( v \) output objects \( y \in P^v \). We denote the set of all process steps as \( Q \). Please note that parameters of process steps can be implemented by defining several individual process steps.

Based on these application scenarios, a common causality model for the AI reference model AITwin is derived. Figure 5 shows two types of causalities, namely System State Causalities (handling system changes) and Product State Causalities (handling product changes) are defined:

System State Causalities: Such causalities model the transition from a system state \( s_i \in S \) to state \( s_j \in S \). The occurrence of an event \( e_3 \) triggers, denoted by Pearl’s \( \text{do} \) operator [24], this transition. Please note that also empty events, i.e. immediate transitions without external triggering events, are possible.

A system state \( s \in S \) corresponds to a subset of \( \mathcal{X} \), with \( \mathcal{X} \) being the set of all possible values \( x(t) \). Here we restrict this to convex subsets of \( \mathcal{X} \), corresponding to linear inequalities on \( x(t) \). I.e. a system state is defined as a subset of the values.
An event $e_1$ is then defined as the crossing of a threshold in $\mathcal{X}$. Such thresholds correspond to equations of the form $f \cdot x(t) < c, f, x(t), c \in \mathbb{R}^n$ which define a halfspace of $\mathcal{X}$. Such a system state causality is only valid under the condition that specific components $C_1, \ldots, C_l$ are functioning correctly, e.g. $OK(C_1) \land \ldots \land OK(C_l)$ is true (see section IV-B). Please note that with these information the diagnosis task from above can be implemented, e.g. consistencies between predictions $x(t)$ from the predictions API level and causalities predicting system states $s_i$ can be checked as $s_i$ is a subspace of $\mathcal{X}$.

**Product State Causalities:** Such causalities model the transition from one set of objects $p_1, \ldots, p_j$ to another set $p_k, \ldots, p_l$. A product $p_k$ might be any product, partial product or raw material, e.g. a chemical, a half-finished car or a complete car.

Again, the occurrence of an event $e_2$ triggers this transition and empty events are possible. Events are defined as above. Such a system state causality is only valid under the condition that specific components $C_1, \ldots, C_l$, see section IV-B, are functioning correctly, e.g. $OK(C_1') \land \ldots \land OK(C_l')$ is true. The resulting API extensions are shown in figure 6. Please note that with these information the planning task from above can be implemented.

An implication API allows for the handling of causalities in form of transitions or implications: The function getSystemCausalities($s_i$) returns all causalities $s_1, e, s_2, i, n, OK$. $s_1$ denotes the starting state, $s_2$ the end state, $e$ is the event, $n$ the name of the causality and $I$ stores additional information such as probabilities, timing etc. $OK$ is the set of $OK$ assumptions $OK = OK(C_1) \land \ldots \land OK(C_l)$ which are a precondition for the validity of the causality.

The function getProductCausalities($p_l$) returns all causalities $p_1 = (p_{11}, \ldots, p_{1j}), e, p_2 = (p_{21}, \ldots, p_{2j}), i, n, OK$ where $p_1$ denotes the original objects, $e$ is the event, $p_2$ denotes the resulting objects, $n, I, OK$ are defined as above.

$e = \text{defineEvent(halfspace)}$ defines an event $e$ which corresponds to a halfspace. The parameter halfspace is defined using a mathematical inequality in the language MathML [21]. The function getEvent($x, x'$) returns all events which are triggered when moving from the operation point $x \in \mathcal{X}$ to $x' \in \mathcal{X}$.

![System State Causalities](image1)

![Product State Causalities](image2)

**Fig. 5.** The main causality concept of the DigitalTwin AI reference model AITwin. $s_i, s_j$ are system states, $C_j$ are components, $p_k$ are products.

**Fig. 6.** The complete DigitalTwin AI reference model AITwin.

**D. The AITwin Reference Model**

Figure 6 shows the complete DigitalTwin AI reference model AITwin. Please note that the focus is on those parts of a DigitalTwin which are required by AI algorithms. Further information for domains such as production, automation and logistics must be added. AITwin may be used in parallel on different system granularities, e.g. for machine modules in production plants or for production lines.

Please note that the three levels of figure 2 correspond to the three levels of figure 6 and form a holistic concept: Data $x$ from the "Synchronized Data Model"-API is used also in the "Prediction-enabled Closed-Loop Model"-API. And these data points can also be correlated to system states from that "Causality Model"-API. And all APIs use the same components for modeling failure modes.

Besides those APIs, the DigitalTwin is based on three state spaces which must be part of the DigitalTwin definition: $\mathcal{X}'$ denotes all possible signal values, $P$ denotes all possible object/product configurations and COMPS is the set of components which may fail. Furthermore, two derived state spaces exist, which normally are defined via API methods during the operation phase: $S$ is a set of system states where $s \in S$ is a subset of $\mathcal{X}'$ and $E$ is the set of all events where $e \in E$ is a halfspace of $\mathcal{X}$.

Please note that the APIs here do not cover the way how the information are added to the DigitalTwin.

**V. EVALUATION**

Two ways exist to evaluate such a new API. In section V-A, AITwin is used by several AI algorithms for one running example. In section V-B, previous research projects in the field of AI in production are analyzed whether AITwin would have been suitable.
### TABLE I

RESULTS OF THE MACHINE LEARNING METHODS

<table>
<thead>
<tr>
<th>Config.</th>
<th>Fault</th>
<th>LSTM Autoen</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>AUC</td>
<td>F1</td>
</tr>
<tr>
<td>a_tank</td>
<td>stable</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>a_tank</td>
<td>pumpSlow</td>
<td>0.500</td>
<td>0.406</td>
</tr>
<tr>
<td>a_tank</td>
<td>tank1_leak</td>
<td>0.500</td>
<td>0.402</td>
</tr>
<tr>
<td>a_tank</td>
<td>valveBBlock</td>
<td>0.999</td>
<td>0.977</td>
</tr>
<tr>
<td>a_tank</td>
<td>valveBBlock</td>
<td>0.994</td>
<td>0.950</td>
</tr>
<tr>
<td>a_tank</td>
<td>valve3Stuck</td>
<td>0.999</td>
<td>0.977</td>
</tr>
<tr>
<td>4_tanks</td>
<td>stable</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>4_tanks</td>
<td>pipe4_jam</td>
<td>0.996</td>
<td>0.996</td>
</tr>
<tr>
<td>4_tanks</td>
<td>tank2_leak</td>
<td>0.960</td>
<td>0.959</td>
</tr>
<tr>
<td>4_tanks</td>
<td>valve3_jam</td>
<td>0.996</td>
<td>0.996</td>
</tr>
<tr>
<td>4_tanks</td>
<td>valve5_jam</td>
<td>0.996</td>
<td>0.996</td>
</tr>
<tr>
<td>3_tanks</td>
<td>stable</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>3_tanks</td>
<td>pumpFast</td>
<td>0.915</td>
<td>0.833</td>
</tr>
<tr>
<td>3_tanks</td>
<td>pumpSlow</td>
<td>0.993</td>
<td>0.990</td>
</tr>
<tr>
<td>3_tanks</td>
<td>tank1Leak</td>
<td>0.994</td>
<td>0.992</td>
</tr>
<tr>
<td>3_tanks</td>
<td>tank2Leak</td>
<td>0.994</td>
<td>0.992</td>
</tr>
<tr>
<td>3_tanks</td>
<td>valve2Closed</td>
<td>0.988</td>
<td>0.978</td>
</tr>
<tr>
<td>3_tanks</td>
<td>valve3Closed</td>
<td>0.986</td>
<td>0.975</td>
</tr>
</tbody>
</table>

A. Applying AITwin to a Running Example

In a next step, the "synchronized data" and especially the "extrapolation" APIs have been used to detect anomalies. Anomalies can be inserted via the "failure mode" API.

For this, first data is extracted via the "data" API and used to train a model of the normal behavior via unsupervised learning. Later, non-normal data is fed to the models and the models are supposed to detect these anomalies. As learning algorithms, Support Vector Machines (SVM) and Long-Short-Term-Memory (LSTM) neural network are used. Table I shows the results for different systems and different faults. Both Area-Under-Curve (AUC) and also F1 measures show that anomalies are detected with a high probability.

In a second step, the AITwin is applied to symbolic AI tasks. For this, concepts are extracted from the DigitalTwin via the "concept" API and causalities via the "implication" API. This information can be used to create a model for Consistency-Based Diagnoses (see section IV-C). Failures can be added via the "failure" API. Table shows the results, most failures can be identified correctly.

We also used a quantitative simulation of the Tennessee Eastman process. The implementation of Downs et al. [9] contains 20 different injected faults (process disturbances). However, the instrumentation of the simulated process is such that not all faults will be identified exactly. Table II shows the results for six experiments. The injected faults for IDV 16 through 20 have an insufficient description for validating results. The other faults not described contain fault modes that are not associated with single components and thus cannot be evaluated. This leaves us with the experiments shown in Table II.

An ok indicates that the faulty component was found as part of the minimal cardinality diagnosis and a not ok shows that the faulty component was not found as part of the diagnosis. We were able to find all faults that had an identifiable component fault as the cause of the process disturbance. The change of input ratios can only be detected indirectly, since no observations are available at the inputs.

### TABLE II

EXPERIMENTAL RESULTS WITH THE TENNESSEE EASTMAN PROCESS

<table>
<thead>
<tr>
<th>IDV</th>
<th>Fault isolated</th>
<th>Injected Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>not ok</td>
<td>Feed ratio changed</td>
</tr>
<tr>
<td>6</td>
<td>ok</td>
<td>Pipe A feed loss</td>
</tr>
<tr>
<td>8</td>
<td>ok</td>
<td>Feed ratio changed</td>
</tr>
<tr>
<td>13</td>
<td>not ok</td>
<td>Reactor kinetics fault</td>
</tr>
<tr>
<td>14</td>
<td>ok</td>
<td>Reactor cooling fault</td>
</tr>
<tr>
<td>15</td>
<td>ok</td>
<td>Condenser cooling fault</td>
</tr>
</tbody>
</table>

These two result tables show clearly that the AITWin concept can be used for a wide range of AI applications—from data-based machine learning problems to symbolic AI diagnosis problems.

### B. AITwin and Research Projects

In the following, some projects and publications from the field of "AI for Production" are listed and analyzed with regard to the AITwin. For this we split those references into the field of condition-monitoring, predictive maintenance, optimization and the field diagnosis and planning.

### C. Condition Monitoring, Predictive Maintenance, Optimization

There is a huge number of projects and research works in the field of machine learning regarding anomaly detection, condition monitoring, predictive maintenance and optimization and the following review can only cover a small part of this work. Starting with the project SMARTPas [31], which deals with the prediction of beverage characteristics, the data & prediction API can be used to predict new data points for control and optimization. The projects IMPROVE [32] and PrognoseBrain [33], dealing with condition-monitoring and anomaly detection in production systems with ML, could have used the data & prediction API as well. In terms of research works e.g. Li [17], who predicts the surface roughness in extrusion-based additive manufacturing could have used the proposed DigitalTwin. Eiteeneuer [12], performing an anomalous detection with LSTMs, can use the data & prediction API as well. Other examples in which the DigitalTwin could have been used can be found in [35], giving a comparative survey with ML in manufacturing, in [7] using ML in digital manufacturing or in [6] who uses an on-line defect recognition. Within all these projects and research work, instead of having several incompatible data models, possibly resulting in not comparable results, one could have used a single DigitalTwin and the data & prediction API.

### D. Diagnosis and Planning

This section reviews several typical projects and research work in the field of symbolic AI, mainly covering diagnosis and planning tasks. Especially Diagnosis algorithms can, if consistency-based, get all necessary information from the AITwin reference models. It needs to be kept in mind that other diagnosis methods can be added, but within the AITwin concept consistency-based diagnosis is especially suited for CPS—please refer to [8] for details. Regarding the projects,
e.g. EfA [5], which deals with the development of intelligent assistance systems, can use the causality API for the planning of automation systems. The project IMPROVE [32], in which innovative modelling approaches for production systems are developed, can use the causality API for the detection of system causalities. In terms of recent publications, e.g. [27], aiming to do an automated process planning for CPS and [25] doing non-convex hull based anomaly detection in cyber-physical production systems could use the AITwin. Other examples are [1] who gives a detailed review of ML for diagnoses in production or [36] who analysis gas wells. In every work the causality API of the AITwin can be used. Regarding planning tasks, projects and research work can also derive most of its needed information from the AITwin. So far, descriptions of machines must be implemented individually but can refer to process step descriptions as capability models. Furthermore, the computed overall production process is also stored individually, but could be added in a straight forward manner. The bottom-line is that the analyzed consistency-based diagnosis activities for CPS can be implemented with the AITwin model. The planning activities can to a large extent be implemented with the AITwin model, some proprietary information must be added.

VI. CONCLUSION

In section I, three research questions have been identified as key challenges for the definition of a common DigitalTwin reference model, called AITwin. For this, section IV analyzes conceptual requirements of CPS to timing models, section IV-B analyzes conceptual requirements of CPS to machine learning models and section IV-C analyzes conceptual requirements of CPS to symbolic AI models. Based on these requirements, concepts and APIs are derived in section IV-D. This AITwin answers the three research questions.

Section V uses a running example and projects to empirically verify the AITwin reference model leading to the point that most AI and ML activities can be implemented using it. The authors do not claim that the reference models is the ultimate definition of a common AI DigitalTwin. But it is to a significant extent able to meet the requirements and may serve as a crystallization point for further research.

Assuming that both DigitalTwins and AI/ML methods are crucial elements of future CPS architectures, which is a rather safe bet, both worlds need to be harmonized. The AITwin reference model can prevent unnecessary redundant modeling efforts and allows for a reuse and sharing of computed results. Finally, this can be a starting point for a standardization of an AI-enabled DigitalTwin for CPS.

Acknowledgements: This work was partly developed within the Fraunhofer Cluster of Excellence "Cognitive Internet Technologies".

REFERENCES


