# A Model-Based Approach for Monitoring and Diagnosing Digital Twin Discrepancies

**Elaheh Hosseinkhani**
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

**Martin Leucker**
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

**Martin Sachenbacher**[1]
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

**Hendrik Streichhahn**
Dräger Safety AG & Co. KGaA., Lübeck, Germany

**Lars B. Vosteen**[1]
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Germany

## Abstract

Recent decades have seen the increasing use of Digital Twins (DTs) – that is, digital models used over the lifetime of a physical product or system for tasks such as predictive maintenance or optimization – in a number of domains such as buildings, manufacturing, or design. DTs face a challenge known as the *DT synchronization problem*; a DT, often based on machine-learned, or complex simulation models, needs to adequately mirror the physical product or system at all times, as any deviations might affect the quality of predictions or control actions. In this paper, we present a model-based approach that aims to add a level of awareness to DT models by supervising if they are in sync with the physical counterpart. The approach is agnostic to the type of models used in the DT, as long as they are compositional, and based on monitoring critical properties (behavioral or functional aspects) of the system at run-time. In the case violations are detected, it reasons on the DT's structure to localize and identify parts of the model that cause deviations and need to be adapted. We give a formal description and an implementation of this approach, and illustrate it with an example from building climatisation.

**2012 ACM Subject Classification** Computer systems organization → Reliability

**Keywords and phrases** Digital Twins, Runtime Verification, Diagnosis, FDIR, TeSSLa

**Digital Object Identifier** 10.4230/OASIcs.DX.2024.2

**Supplementary Material** *Software (Source Code)*: `https://github.com/vosteen/FDIR` [36]
archived at `swh:1:dir:1972d8d2373f5d9c6e127fa16aec1a79755b056a`

**Funding** This paper is partially funded by the German Federal Ministry for Economic Affairs and Climate Action, due to a resolution of the German Bundestag, under grant number 01MC22005C.

**Acknowledgements** We want to thank Jan-Henrik Metsch for insightful discussions and valuable feedback.

## 1 Introduction

In recent decades, steady progress in areas like the Internet of Things, Cyber-physical systems, Model-based engineering and Artificial Intelligence, and Cloud computing has enabled the digitalization of different assets such as products, systems, and processes in

[1] corresponding authors

35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024).
Editors: Ingo Pill, Avraham Natan, and Franz Wotawa; Article No. 2; pp. 2:1–2:15
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

different industrial sectors. Specifically, Digital Twins (DT) have become a popular concept in various application fields such as urban construction, manufacturing, automotive systems, or individualized healthcare [28, 29, 37, 26]. A DT is a set of adaptive models that emulate the behaviour of a physical asset (termed Physical Twin, or PT) in a virtual system, getting real time data to update itself along its life cycle. The DT replicates its physical counterpart to enable and analytically improve tasks like what-if analysis, monitoring and predictive maintenance, or optimisation and parameter tuning based on the communicated data. The practical relevance is underlined by the existence of various (commercial) DT tools and frameworks such as Microsoft Azure's Digital Twins and its corresponding Digital Twins Description Language (DTDL) [22], Plattform Industrie 4.0's Asset Administration Shell (AAS) [1], and INTO-CPS Digital Twin as a Service [4]. Despite this, DTs have notoriously evaded a more standardized or formally strict definition beyond the basic understanding that there exists a digital representation that is kept synchronized at a specified frequency and fidelity with its physical counterpart through communication channels (for instance, see the set of different definitions resulting from a recent Dagstuhl seminar on the topic [10]). This may be attributed to the multi-disciplinary nature of the concept and to the quite broad understanding of digital representations (models), which might range from light-weight visualizations to logical and ontological representations, to machine-learned models and complex numerical simulation models.

Shared by all instances of DT technology, however, is the key challenge that the DT's model needs to adequately mirror the physical product or system, as any deviation might deteriorate the quality of the obtained predictions, alerts, or actions. This problem is often referred to as *DT synchronization problem* ([33, 11]). We use the same term here, but want to note that while the literature emphasizes the role of the communication link between the DT and the PT, we view synchronization more broadly as the problem that the DT needs to faithfully represent the behavior of the PT. As the DT is meant to match the PT along different stages of its entire life cycle, synchronization is a continuous effort and there are potentially several reasons which might cause the DT to be out of sync. They could be roughly classified into failures (component breakdowns) in the PT, parameter drifts e.g. due to wear in the PT, or communication faults between the DT and the PT.

Although various solutions have to date been proposed to the DT synchronization problem, they are typically limited to specific contexts. For instance, Kamburjan et al. [16] propose a solution that exploits ontology models (OWL) and domain-specific constraints to dynamically reconfigure simulators in a DT. Tan and Matta [33] give an overview of different types of DT syncronization problems occuring in the context of simulation and a solution when to best update simulation models based on a cost model.

Of course, the fields of Model-based Diagnosis (MBD) and Fault Detection, Identification and Reconfiguration (FDIR) offer a rich and long-standing body of research about how to derive and maintain model representations that truthfully represent a physical artifact's state, referred to as mode estimation or state estimation, respectively (see e.g. [12, 24, 34]).

However, these approaches are based on specific classes of models (structural or logical models, discrete-event automata, residual equations, etc.), as not all forms of models (e.g., machine-learned or simulation models) are directly amenable to diagnostic reasoning, which requires tracing the system backwards from observed symptoms to inner component states. Actually, in meaningful DT applications the models can be quite large (e.g., city-scale) and heterogeneous (e.g., composed of different simulation units requiring precise timing and orchestration), making first-principles diagnostic and causal reasoning difficult. There also exist methods aiming to transform some types of simulation models (such as Modelica models) into diagnostic models [23, 27, 18], which could work to address some forms of DT synchronization issues. Also more recently, MBD methods have been used to update/repair specific forms of learned models in the face of additional data [2].

In this paper, we present a different model-based approach to the problem of DT synchronization and fidelity during runtime. Our approach is agnostic to the specific type of DT models, using only structural information (assuming a compositional DT) to be as general as possible. To capture behavioral and functional aspects of the DT that are critical to synchronization, we instead rely on a set of specifications expressed formally in some temporal logic. Our approach then consists of monitoring these properties during run-time using techniques from Runtime Verification (RV). Once violations of properties are detected, MBD on the structural model is used to identify root causes of violations. An approach for coupling RV and MBD has already been presented in previous work by Bauer, Leucker and Schallhart [6]. It is based on structural models and focuses on fault detection and localization in distributed systems as part of a runtime reflection framework. We build on this approach, and adapt it to the context of DTs. Also, we identify several theoretical and practical challenges that arise when coupling RV and MBD in DTs. Using RV for discovering divergences between DTs and their real-world counterpart has also recently been suggested in [13]. However, this work focuses more on conceptual variants and challenges for RV, such as proper specification languages to describe meaningful divergences. In contrast, we present an actual implementation of coupling RV and MBD for DT, and illustrate it with a concrete example.

The remainder of the paper is structured as follows. In the next section, we review relevant background on DTs and RV using monitors. We then present our approach of coupling RV and MBD to identify and localize discrepancies between a DT and its PT, in order to keep the DT in sync. Our prototypic implementation is based on DTDL+, an extension of Azure's DTDL as an executable DT specification, and the temporal stream-based specification language TeSSLa [15] for RV and monitoring. We present experimental results for a small example of a DT for building climatization.

## 2 Background

### 2.1 Digital Twins

Digital Twins (DTs) are virtual models that represent real-world entities and systems. They are commonly used for predictive purposes, allowing the simulation of various potential outcomes of the physical counterpart. For a DT to be effective, it must accurately reflect the behavior of the real-world system it represents. Any significant or persistent divergence between the DT and its physical counterpart can result in flawed predictions, incorrect diagnoses, and a distorted understanding of the system's operation. Therefore, discrepancies between the DT and the real-world entity must be continuously monitored and promptly addressed in real time.

Figure 1 illustrates the structure of a DT. The physical entity, referred to as the PT, receives inputs ($I$), which may be either controllable (user-defined) or uncontrollable (environmental) factors. The PT responds to these inputs by producing outputs ($O_p$), which could either be actions performed by the PT or measurements of its state or environment. Simultaneously, the inputs are also provided to the DT, which simulates the behavior of the PT and generates its own outputs ($O_d$). As shown by the dotted lines in Fig. 1, certain outputs from the DT ($O_d$) may be used to control the PT, while some outputs from the PT ($O_p$) are fed back to the DT as observational data.

Although sometimes (e.g. Kritzinger et al. [17]) different types of DTs – namely, digital models, digital shadows, and DTs – are further distinguished based on the direction of communication between the PT and DT (one-way or two-way), for our purposes we do not distinguish between these variations.

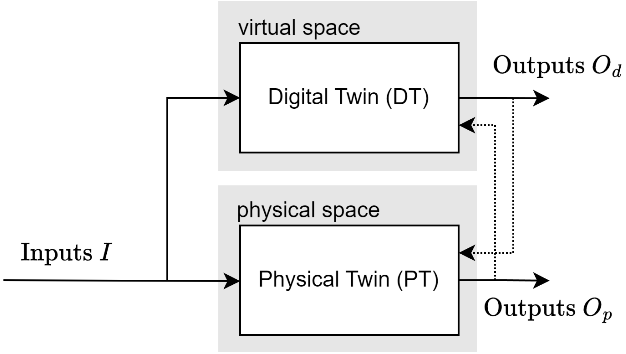

**Figure 1** Schematics of a Digital Twin.

In [19], we identified three different artifacts as the key ingredients of a DT's formal representation. These artifacts are

- *environmental models*, such as building models, that provide background knowledge about the structure (e.g. floor plan) and semantics (e.g. accessible doors and stairs in a building) of the DT's context, to represent e.g. locations of people and plan e.g. feasible routes;
- *communication and discrete optimisation*, such as updating the people's positions and finding the shortest path in a building and travelling salesman to search all rooms;
- *physical process models*, that allow to simulate e.g. how long the air supply of a firefighter will last along a route, or the thermal dynamics of heat dispersion in a building.

Like in other fields of model-based engineering, digital twinning incorporates the idea that a DT should be compositional, i.e. composed of (reusable) blocks corresponding e.g. to simulation units or FMUs. For example, in DTDL [22], any DT instance consists of the classes Interface, Component, Command, Property, Relationship, and Telemetry. We use the compositionality of DTs in our approach by exploiting the structure of DTs to reason about possible causes of discrepancies.

## 2.2 Runtime Verification and Monitoring

RV [20, 5] is a formal method from the area of software engineering. In RV, objects called monitors are used to observe the behavior of programs. The data stream of interest is transmitted to the monitor, producing a sequence of events called traces. A monitor compares this event trace against a formal specification $\varphi$ that holds in a correct execution. The monitor can report violation or validation of the desired specification. Its verdict is reported on-the-fly, during the execution of the monitored program. Other than testing, RV is a passive procedure that does not actively generate inputs to drive the system. Compared to model checking, which is concerned with mathematically proving properties that must hold in all execution of the system, RV is more lightweight and only concerned with concrete traces of the running system.

Research in RV has produced a variety of monitors that can be automatically generated from specifications $\varphi$ formulated in variants of linear time temporal logic (LTL), and corresponding higher-level specification languages such as LOLA [3] or TeSSLa [15]. However, as a monitor can have only a finite view (trace) of the system's behavior, whereas LTL is defined over infinite traces, a semantics has to be defined for RV on finite traces. In [7], a three-valued semantics is proposed for this case, where the monitors can either report $true$, $false$, or ?, the latter meaning an inconclusive verdict, indicating that the trace observed so far does not allow to decide whether $\varphi$ holds or whether it will be violated in the future.

## 2.3 Property-based Monitoring of Discrepancies

To discover possible divergences between the DT and its physical counterpart, we have to compare their behavior. One way, as also suggested in [13], is to use conditions $\varphi_1, \ldots, \varphi_n$ which must hold for all executions of the DT. These properties, acting as a form of guarantee, might either be given as a formal specification, deductively extracted from a white-box twin beforehand, or generated by observing multiple executions of the (black-box) twin and learning a specification of its behavior. In any case, we suppose the existence of a set of such conditions, $\varphi_1, \ldots, \varphi_n$, beforehand. The properties can be converted into monitors that then observe the operation of the real-world entity in real time. Any observed sequence that violates one of the properties $\varphi_i$ is a sequence that cannot be produced by the DT for the same inputs, and thus indicates a divergence between the DT and its real-world counterpart.

## 2.4 Diagnostic Reasoning from Property Violations

MBD and FDIR deal with the problem of detecting if a system is not functioning correctly, and determining which part of the system is failing (and which kind of fault is present). It is based on reasoning from observations, which provide information on the actual system's behaviour, and in particular discrepancies between values predicted by a model and these actual observations. These discrepancies (called symptoms) are the starting point for diagnosis and the goal is to explain them by hypothesizing (a minimal number of) faults in the system that cause the deviation between predicted and actual behavior.

If properties of the system are monitored, as outlined in the sections above, their fulfillment can be used in lieu of the observations. They can be viewed as adding an additional layer of abstraction: instead of considering discrepancies of values in the system, the starting point for diagnosis are violations of properties by the system. The "classic" notion of symptoms, i.e. discrepancies between predicted and observed values, can be re-gained in this case as a special case of (violated) properties that demand equality of two variables (predicted versus actual). In this sense, monitoring of properties is a powerful concept that allows to capture complex symptoms, involving (temporal) behavior of variables.

However, for diagnosis, this creates the challenge how such generalized "symptoms" – violation of one or more properties on the system's trace – could be attributed to parts of the system failing, and the type of fault(s) occuring. It is not obvious how faults of system components (whose behavior is modelled as propositional logic sentences, finite state machines, etc.) could be related to violation of (arbitrary) temporal logic properties.

To date, a number of approaches have been suggested for coupling run-time property monitoring and MBD. [6] use an approach where the compositional models abstract away from the component's behavior. More precisely, the component models state that if all the inputs of a non-abnormal component are correct, then its output must also be correct, allowing one to infer that it cannot be responsible for violating a property monitored downstream at its output. Though a coarse abstraction, such "behavior-free" models can be used to compute conflicts and diagnoses, and they can be easily derived from the graph structure of the model. Diagnostic models based on propagating deviations [32, 31], where symptoms correspond to values being either too low, too high or ok, can also be viewed as a form of diagnostic reasoning with property violations, although the properties themselves are not stated explicitly. In [25], the authors present an approach for diagnosis of multi-agent systems where the agents and a system goal are observed across multiple executions (runs), and a spectrum-based fault fault localization (SFL) method is used to explain failed runs by diagnosing which agents are faulty. As it defines the success and failure of an execution based on a property of the system rather than states of the components, this can also been seen as a form of property-based monitoring approach that we aim to formalize in this paper. However, while [25] assumes the evaluation of runs at the very last time step after all executions are finished, our approach uses RV as a fault detection mechanism that can also work on-line.

In the field of RV, assumption-based RV has been proposed in [9] as a framework that leverages assumptions on the behaviors of the system under scrutiny for reasoning on its non-observable (internal) or future behaviors. The idea is that the monitor outputs (property satisfied or not satisfied) are conditioned on the assumption that the involved parts of the system are behaving correctly, allowing one to reason about these assumptions in a model-checking framework.

## 3 Building Climatization Example

In an on-going project[1], novel approaches for designing, deploying, and monitoring DTs are developed in the context of buildings and emergency scenarios. Specifically, to enhance the DT model's accuracy and longevity, the aim is to add awareness regarding some features and to supervise the DT's performance through monitoring and fault localization. Loosely derived from this project, we present a simplified example to motivate and illustrate our approach[2]. The example system consists of two parts, the PT and the DT:

*Physical Twin.* Consider a makeshift building (tent) separated into three heating zones. Each zone has a heater ($H_1, H_2$, and $H_3$), controlled with an on-off power value $h_i \in \{0, 1\}$. The temperature within the zones is measured by four sensors, $T_A$ to $T_D$, strategically placed at the periphery and between the heating zones. Each sensor provides a measurement $t_i$ in °C. Figure 2 provides a schematic overview.

*Digital Twin.* The primary task of the DT is to maintain the temperature in each zone above 20 °C to ensure occupant comfort, while optimising for energy efficiency by not heating unnecessarily. Each zone has its own heating controller ($C_1$ to $C_3$), each of which sends a control signal. To do this, the temperature propagation in the PT is simulated within the controller to inform the decisions. In the problem classification mentioned in 2.1, this represents a physical simulation and is approximated by an ordinary differential equation that is solved iteratively. The rest of the DT system is based on digital processes.

The second purpose of the DT is to increase reliability. To this end, the monitors $M_A$ to $M_D$ monitor the output values of the temperature sensors $T_A$ to $T_D$ and identify zones where the temperature targets are not being met (significantly below 20 °C) or are not progressing satisfactorily (e.g, the temperature $t$ has not increased by 5 % from the initial value $t_0$ within 42 s). Whenever a monitor is triggered, we want to identify components that may be responsible for the fault. In addition, the heater controllers monitor their internal simulation by keeping track of whether they are providing results that match the measurements from the PT. Whenever a loss of simulation alignment is detected, an automatic synchronisation of the simulation model to better fit reality should be initiated.

[1] `https://o5g-n-iot.de`

[2] The source code of the example is available in the repository `https://github.com/vosteen/FDIR`

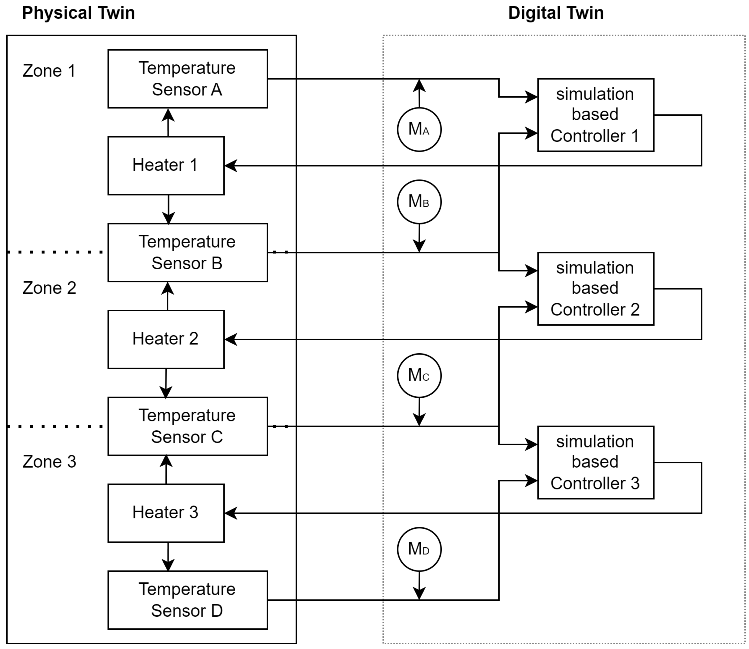

**Figure 2** Schematic overview of the running example.

# 4 General Approach

In this section, we combine the principles laid out in Section 2 to propose an approach for detecting discrepancies between a DT and its PT, i.e. checking the synchronicity of DT, and localizing possible root causes (diagnosis) at runtime.

## 4.1 A Model for Property-based Diagnostic Reasoning

We consider a system (DT) composed of a finite set of components, denoted $COMP$. Diagnosis aims to determine whether a subset of the components is faulty (leading to discrepancies between DT and PT). A special predicate, $AB$, is used to denote that a component is abnormal, i.e. behaving different from its intended or specified behavior.

As usual in MBD, the system is then represented as a tuple $S = (SD, COMP)$, where $COMP$ are the components and $SD$ is the system description, comprised e.g. of a set of first-order sentences (the exact type of model is not relevant in the following). We assume the components are directed, i.e. they have inputs and outputs (for simplicity, we assume a single output).

The main difference of our approach to "classical" MBD is in the definition and use of observations, typically denoted $OBS$. We assume a set of properties $\varphi_1, \ldots, \varphi_n$, denoted $PROP$, which must hold for all executions of the DT. We use the properties – more specifically, the verdicts (outputs) of corresponding monitors constructed from these properties – in lieu of direct observations of the system. This leads to a further abstraction of the system model: instead of observations of actual system values, we only use the information whether the values conform to the specified properties. Since the properties we are after are in general defined on streams of values instead of single values, we refer to "channels" instead of values in our model. Following the same ideas as in [6], we use a predicate $ok$ to denote whether a channel behaves according to the specified properties, and devise a model that only captures the causal dependencies between the channels.

The system description is then reduced to a set of first-order sentences describing the correctness of input-output behavior of the components and channels. That is, for every component $C_j \in COMPS$ with inputs $i_1, \ldots, i_m$ and output $o$ we get a formula

$$ok(i_1) \wedge ok(i_2) \wedge \cdots \wedge ok(i_m) \wedge \neg AB(C_j) \implies ok(o)$$

where *ok* denotes that a value or channel does not violate any specified property, and $\neg AB$ denotes that the component is not abnormal. As described in Section 2.2, the monitors check whether a sequence of events satisfies a certain property $\varphi_k \in PROP$. The monitors can then report either $true$, $false$, or inconclusive (denoted ?). For diagnosis we are interested in finding the causes of violations of properties, and we can thus identify $true$ and ? [6]. Thus, as observations $OBS$ of the system, we get the information whether certain values (channels) in the system are *ok* (if the monitor reports $true$ or ?) or $\neg ok$ (if the monitor reports $false$). More formally, let $v_k$ denote the values (channels) monitored by the properties $\varphi_k$. Then the observations are

$$OBS = \{ok(v_k) \mid \varphi_k = true \vee ?\} \cup \{\neg ok(v_k) \mid \varphi_k = false\}$$

In sum, this gives us the familiar notion of consistency-diagnosis, but with the specific interpretation of $SD$ and $OBS$ outlined above.

## 4.2 Extending the Model to Temporal Property-based Diagnostic Reasoning

The model described in the section above, taken from [6], abstracts away from the components internal behavior, in particular also their temporal behavior. It assumes that the *ok*-predicates for the outputs are generated instantaneously from the *ok*-predicates of the inputs of the components.

Considering the different artifacts of a DT as outlined in Section 2.1, this assumption appears reasonable for the “digital”, i.e. the *communication and discrete optimisation* components of a DT. In fact, if we apply the modeling approach described in Section 4.1 and [6] to the DT scenario from Section 3, including the *physical process models*, a problem appears: as the example contains a physical process (heat dissipation) together with sensing, control and actuating components, a causal loop will be created in the model reasoning about the *ok*-predicates. The consequence (for the example, but also in general) is that the diagnostic model would in this case yield all components as possible diagnoses, not allowing any further discrimination.

Looking more closely, the underlying problem is that the model from Section 4.1 is a too coarse abstraction of the DT; we must at least distinguish the *communication and discrete optimisation* models from the *physical process models*. In this example, switching on a heater component has an influence on the zone’s temperature, but due to thermal inertia, the effect will be delayed and manifest only in a later time step. The interaction between the heater components and the temperature sensors is thus via the thermodynamics of the respective zone, which could be governed by differential equations and in any case happens at a different time scale compared e.g. to the interaction channel between the sensors or controllers and the controllers and the heaters. Only if we model these interactions with underlying physical processes in a more fine-grained way, we can relate the observations from the monitors in the control loop in a way that is meaningful for diagnosis.

In the following, we present a simple extension to the model from Section 4.1 and [6] to adequately deal also with *physical process models* that are present in a DT. For this, we augment the values (channels) $v_i$ in the model with (discrete) time steps $t = t_0, t_1, \ldots$ such that values $v_i^{t_0}, v_i^{t_1}, \ldots$ at different time steps can be distinguished. We are not concerned here with how such time-discrete models are obtained, but note that there are methods to generate them from underlying differential equation models by abstraction, as e.g. in [21].

Using the extension of values to time steps $v_i^t$, we can extend the causal-structural model from Section 4.1 to include also components that relate variables (in particular, the *ok*-predicates of variables) at different time steps (for simplicity, we limit ourselves to consecutive time steps):

$$ok(i_1^t) \wedge ok(i_2^t) \wedge \cdots \wedge ok(i_m^t) \wedge \neg AB(C_j) \implies ok(o^{t+1})$$

For the example, the interaction between the heaters and the temperature sensors (via a physical process) can be captured with such time-augmented formulas, whereas the other components in the example can be modeled as in Section 4.1. Together, we get a diagnostic model of both the DT's *physical process models* and *communication and discrete optimisation* components that describes the causal input-output propagation behavior. In Section 5, we describe how this model and the observations from the monitors can be used to compute diagnoses.

## 4.3 A Formal Digital Twin Description Language

Our diagnostic approach requires two pieces of formal knowledge for a DT: The system description $SD$ with components $COMPS$ and their connecting channels, and the specification of properties $PROP = \varphi_1, \ldots, \varphi_n$ which are then turned into monitors to yield the observations $OBS$. To represent this information about the DT, we developed an extension of the existing DTDL, which we call DTDL+ [30]. While DTDL describes only abstract classes (interfaces) of DT components and their relationships, DTDL+ enriches it with an `Instance` class. This allows to describe concrete instances of DTs or DT components (for instance, several instances of the same component type), as required to specify system descriptions $SD$. The `Instance` class includes also complete deployment information, including package installation, file transfer, service creation, and the type of data transfer to be used in the DT. DTDL+ descriptions can be automatically translated into various specialised forms such as Ansible playbooks, which enable deployment and reconfiguration. The current description focuses primarily on the DT components that drive the use case.

In addition, there are already concrete ideas for the automated integration of further essential elements such as:

- the reality or its simulation,
- the visualisation component,
- the monitors, and
- the identifiers, which can then automatically utilise the DTDL+ description.

The integration of these aspects is currently in progress.

# 5 Implementation and Application to the Example

We have implemented the approach using DTDL+ for the DT model specification, TeSSLa for generating runtime monitors from specifications, a numerical simulation framework, and a SAT solver to compute diagnoses as described in Section 4. The resulting setup for the example is shown in Fig. 3. It consists of a set of components designed to simulate, control and monitor the thermal environment of a building structure. The source code can be found in our repository[3].

[3] `https://github.com/vosteen/FDIR`

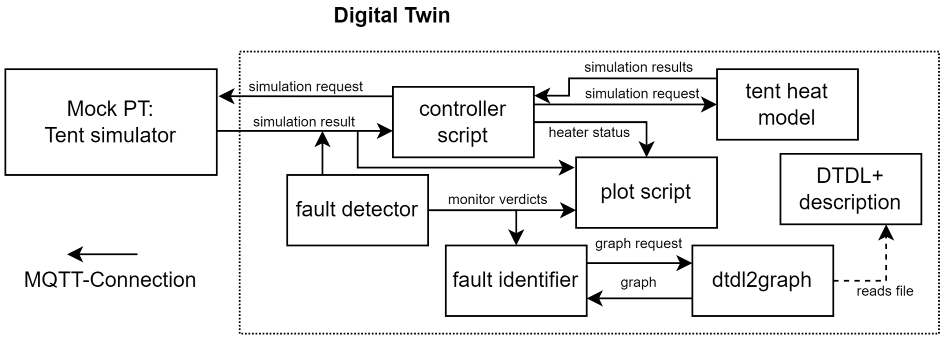

**Figure 3** Experimental setup of the implementation used for the example.

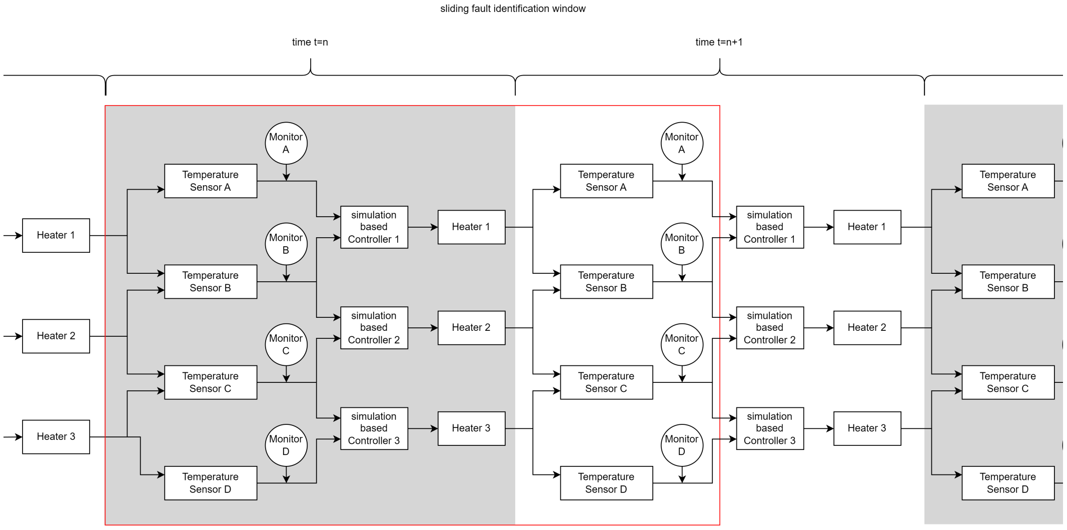

**Figure 4** Diagnostic model of the DT unrolled for a horizon of two time points.The red box presents the subgraph that is used for fault identification.

At the core of the system is the Mock PT, acting as a thermal simulator that processes heating control signals and provides temperature projections for the next time step. This simulation is based on the ISO 52016-1 standard [14], adapted from [35]. The heater controller determines whether to activate a heater based on the digital twin's internal simulation. If the simulated temperature for the controlled zone is predicted to drop below 20 °C, the heater is switched on or remains on for the next cycle. The Mock PT is used to evaluate the fault diagnosis system. The digital twin can be automatically deployed using DTDL+ descriptions. The controller regulates the tent's temperature by switching the heaters. Heater statuses and sensor data are displayed graphically via plot scripts, while DTDL+ descriptions are processed to generate a fault identification graph. Faults detected by the monitoring systems are identified using a SAT-based model. For fault detection, we use a monitor to listen to temperature sensor data streams (see Fig. 2 for an overview). These streams report to the fault identification process.

Towards identifying faults in components, the monitoring component continuously evaluates the temperature data according to the following rule. In TLTL [8] this condition can be formalised as $G_{[t-42\,\mathrm{s},\,t]}\,(\mathrm{temp} < 19.5\,^{\circ}\mathrm{C}) \wedge (\mathrm{temp}(t) \geq 1{,}05 \times \mathrm{temp}\,(t - 42\,\mathrm{s}))$, and is used to generate the respective four monitors $M_a$ to $M_D$. This TLTL condition specifies two possible scenarios: First, that the temperature in the last 42 s ago was greater than 19.5 °C. Second,

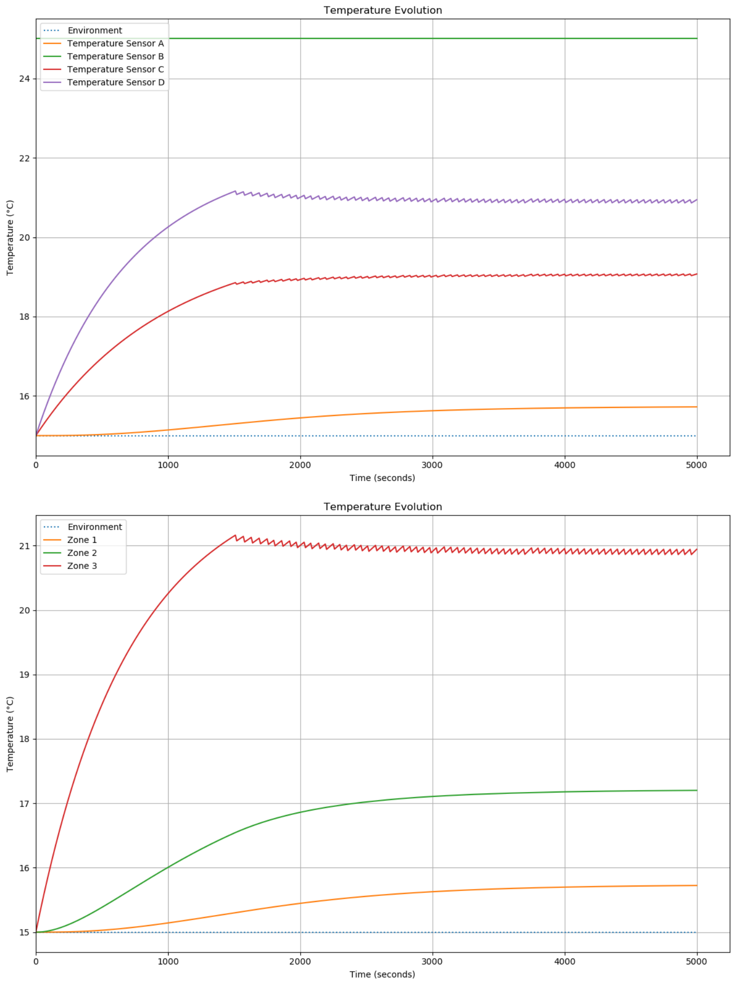

**Figure 5** The plot for the temperature sensor on the top shows the faulty $T_B$ stuck at 25 °C. The heater controllers of Zone 1 and 2 average the temperature signals from $T_B$ and $T_A/T_C$ and see their objective (20 °C) reached. As seen in the plot of the corresponding simulated zone temperatures on the bottom, it is actually not reached.

that the current temperature has increased by at least 5% compared to the temperature from 42 seconds ago. The condition is satisfied if either one of these scenarios is true. When this monitor detects a violation, it triggers the fault identifier, which then uses a diagnosis to analyse the data and determine the possible causes of these anomalies. We use the SAT-based approach proposed by Bauer et al. [6], transforming monitor verdicts and the diagnostic model into a SAT instance. Since our example involves temporally extended component models, we handle temporally indexed variables by considering each component as a separate entity at each time step. To limit the diagnosis problem to a finite instance, we unroll the model over a fixed time horizon. Fig. 4 shows an example unrolled for two time points, which is sufficient for accurate diagnosis in our case. The time horizon can be optimized for faster diagnosis or completeness. The diagnosis provides sets of components that, if abnormal, would explain the monitor's observations. To get an up-to-date component graph, the diagnosis triggers the `dtdl2graph` script to extract one from the DTDL+ description. This is sufficient to create the system description part of the SAT problem from [6].

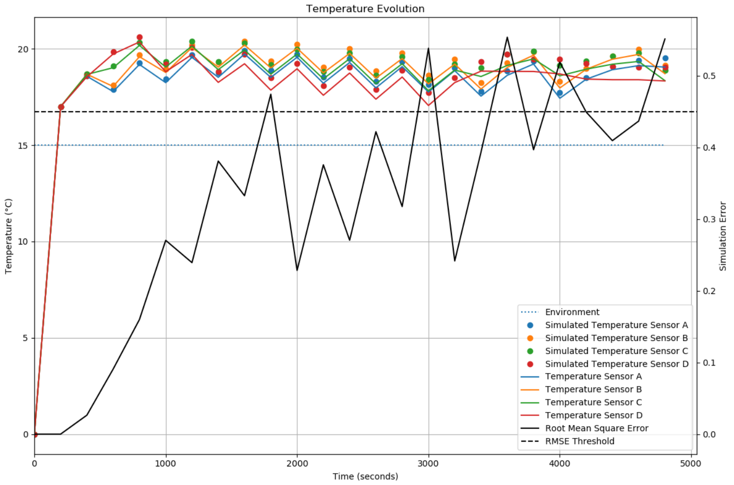

**Figure 6** Example of parameter drift and periodic resychronisation: whenever the difference between the PT and the internal simulation of the controller grows too large (as indicated by the black line crossing the limit shown with the dotted line), the simulation is adapted to better predict reality.

To create a diagnostic scenario, a fault can be injected for one or multiple components in the simulation. We take Temperature Sensor $B$ as an example and configure it to provide the constant and (mostly) false temperature signal 25 °C. Figure 5 shows a temperature plot of this simulation run. The error leads to a violation of the monitors $M_A$ and $M_C$, that is, these temperature values are observed as $\neg ok$. The subset-minimal diagnoses are then determined as follows:

- $T_B$ is abnormal,
- $T_C, T_A$ are both abnormal,
- $H_3, T_A$ are both abnormal,
- ...

The diagnoses could be used as a basis for recovery actions to bring the DT back in sync.

As a scenario for a loss of simulation synchronicity, let's assume the controllers keep the internal simulation results upon which they based their last decisions in memory and compare it to the (simulated) measurements from the PT. If the difference grows too large, the controller-intern monitors detect a fault and trigger a recalibration of the simulation to the measurements. Fig. 6 shows the results of this approach on our example. We then employ a simple gradient descent to find a new set of simulation constants to determine updated (locally) optimal simulation parameters to re-establish synchronicity with the PT.

## 6 Conclusion

DTs are nowadays a well-established engineering concept for model-based design, development, operations and maintenance for Cyber-physical systems. Thus, besides the development phase, they must also support the evolution of the system and track their physical counterparts (PTs) throughout the whole lifecycle, creating the necessity to update the DT when the physical entity changes.

We presented an approach that uses RV to detect if the DT is *out of sync* with its physical counterpart, and uses MBD to localize the reasons for discrepancies of a DT with its PT in an abstracted model representing the causal structure of the DT. For this, we extended an approach from [6] to be able to deal also with physical process models that exhibit dynamic behavior. Our approach also includes the development of DTDL+ as a description language that extends Azure's DTDL to executable formal specifications, and the temporal stream-based specification language TeSSLa for RV and monitoring. An example from a building DT and its prototypical implementation illustrates the effectiveness of the approach. We currently work to expand the approach to more complex examples in the area of DT for emergency mission support [19].

The approach is a step in the ongoing development of resilient DTs, that incorporate a level of self-awareness by automatically monitoring their own performance and fidelity and identifying root causes in the case of critical violations.

## References

1 Plattform Industrie 4.0. Asset Administration Shell. Accessed on August 19, 2024. URL: `https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/AAS-ReadingGuide_202201.pdf?__blob=publicationFile&v=1`.

2 Shaked Almog and Meir Kalech. Diagnosis for Post Concept Drift Decision Trees Repair. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pages 23–33, August 2023. `doi:10.24963/kr.2023/3`.

3 Ben Angelo, Sriram Sankaranarayanan, Cesar Sanchez, and Many Others. Lola: Runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 166–174, July 2005. `doi:10.1109/TIME.2005.26`.

4 INTO-CPS Association. Digital Twin as a Service. Accessed on August 19, 2024. URL: `https://into-cps-association.github.io/DTaaS/`.

5 Ezio Bartocci, Yliès Falcone, Adrian Francalanza, and Giles Reger. *Introduction to Runtime Verification*, pages 1–33. Springer International Publishing, Cham, 2018. `doi:10.1007/978-3-319-75632-5_1`.

6 Andreas Bauer, Martin Leucker, and Christian Schallhart. Model-based runtime analysis of distributed reactive systems. In *Proceedings of the Australian Software Engineering Conference (ASWEC'06)*, page 243–252. IEEE, IEEE, 2006.

7 Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Tasiran, editors, *Runtime Verification, 7th International Workshop, RV 2007, Vancouver, Canada, March 13, 2007, Revised Selected Papers*, volume 4839 of *Lecture Notes in Computer Science*, pages 126–138. Springer, 2007. `doi:10.1007/978-3-540-77395-5_11`.

8 Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4), September 2011. `doi:10.1145/2000799.2000800`.

9 Alessandro Cimatti, Chun Tian, and Stefano Tonetta. Assumption-based runtime verification with partial observability and resets. In Bernd Finkbeiner and Leonardo Mariani, editors, *Runtime Verification*, pages 165–184, Cham, 2019. Springer International Publishing. `doi:10.1007/978-3-030-32079-9_10`.

10 Loek Cleophas, Thomas Godfrey, Djamel Eddine Khelladi, Daniel Lehner, Benoit Combemale, Bernhard Rumpe, and Steffen Zschaler. Model-Driven Engineering of Digital Twins (Dagstuhl Seminar 22362). *Dagstuhl Reports*, 12(9):20–40, 2023. `doi:10.4230/DagRep.12.9.20`.

11 Digital Twin Consortium. The DTC glossary. Accessed on August 19, 2024. URL: `https://www.digitaltwinconsortium.org/glossary/`.

12 Johan de Kleer and James Kurien. Fundamentals of model-based diagnosis. *IFAC Proceedings Volumes*, 36(5):25–36, 2003. 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2003, Washington DC, 9-11 June 1997. `doi:10.1016/S1474-6670(17)36467-4`.

13 Sylvain Hallé, Chukri Soueidi, and Yliès Falcone. Leveraging runtime verification for the monitoring of digital twins. In Stefan Hallerstede and Eduard Kamburjan, editors, *Proceedings of the Workshop on Applications of Formal Methods and Digital Twins co-located with 25th International Symposium on Formal Methods (FM 2023), Lübeck, Germany, March 06, 2023*, volume 3507 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3507/paper1.pdf.

14 Energy performance of buildings – Energy needs for heating and cooling, internal temperatures and sensible and latent heat loads. Standard, ISO, 2017.

15 Hannes Kallwies, Martin Leucker, Malte Schmitz, Albert Schulz, Daniel Thoma, and Alexander Weiss. TeSSLa – an ecosystem for runtime verification. In Thao Dang and Volker Stolz, editors, *Runtime Verification*, pages 314–324, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-031-17196-3_20.

16 Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, S. Lizeth Tapia Tarifa, David Cameron, and Einar Broch Johnsen. Digital twin reconfiguration using asset models. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Practice*, pages 71–88, Cham, 2022. Springer Nature Switzerland.

17 Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018. doi:10.1016/j.ifacol.2018.08.474.

18 Mattias Krysander, Erik Frisk, Ingela Lind, and YIva Nilsson. Diagnosis analysis of modelica models. *IFAC-PapersOnLine*, 51(24):153–159, 2018. 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018. doi:10.1016/j.ifacol.2018.09.549.

19 Martin Leucker, Martin Sachenbacher, and Lars Bernd Vosteen. Digital twin for rescue missions - a case study. In Stefan Hallerstede and Eduard Kamburjan, editors, *Proceedings of the Workshop on Applications of Formal Methods and Digital Twins co-located with 25th International Symposium on Formal Methods (FM 2023), Lübeck, Germany, March 06, 2023*, volume 3507 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3507/paper4.pdf.

20 Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009. The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOSâ'07). doi:10.1016/j.jlap.2008.08.004.

21 Jan Lunze and Jörg Raisch. Discrete models for hybrid systems. In Sebastian Engell, Goran Frehse, and Eckehard Schnieder, editors, *Modelling, Analysis, and Design of Hybrid Systems*, pages 67–80, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

22 Microsoft. Azure Digital Twins. Accessed on August 19, 2024. URL: https://azure.microsoft.com/de-de/products/digital-twins/.

23 Raj Minhas, Johan Kleer, Ion Matei, and Bhaskar Saha. Using fault augmented modelica models for diagnostics. In *10th International Modelica Conference*, 2014. doi:10.3384/ecp14096437.

24 Igor Mozetič. Model-based diagnosis: An overview. In Vladimír Mřrík, Olga Štěpánková, and Rorbert Trappl, editors, *Advanced Topics in Artificial Intelligence*, pages 419–430, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

25 Avraham Natan, Meir Kalech, and Roman BartÃ¡k. Diagnosis of intermittent faults in multi-agent systems: An SFL approach. *Artificial Intelligence*, 324:103994, 2023. doi:10.1016/j.artint.2023.103994.

26 Samuel D. Okegbile, Jun Cai, Dusit Niyato, and Changyan Yi. Human digital twin for personalized healthcare: Vision, architecture and future directions. *IEEE Netw.*, 37(2):262–269, 2023. Conference Name: IEEE Network. doi:10.1109/MNET.118.2200071.

27 Bernhard Josef Peischl, Ingo Pill Pill, and Franz Wotawa. Abductive diagnosis based on modelica models. In *27th International Workshop on Principles of Diagnosis*, 2016.

28 Dimitrios Piromalis and Antreas Kantaros. Digital twins in the automotive industry: The road toward physical-digital convergence. *Applied System Innovations Using Recent Communications and Networking Advances*, 5(4):65, 2022. doi:10.3390/asi5040065.

29 Gerhard Schrotter and Christian Hürzeler. The digital twin of the city of zurich for urban planning. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 88(1):99–112, 2020. doi:10.1007/s41064-020-00092-2.

30 Hendrik Streichhahn. Specification of digital twins for a practicable implementation. Master's thesis, Institute for Software and Programming Languages, University of Lübeck, Lübeck, Germany, October 2023. Supported by Lars B. Vosteen.

31 Peter Struss. Deviation models revisited. In *Working Papers of the 18th International Workshop on Qualitative Reasoning*, 2004.

32 Peter Struss, Martin Sachenbacher, and Florian Dummert. Diagnosing a dynamic system with (almost) no observations. In *Workshop Notes of the 11th International Workshop on Qualitative Reasoning*, 1997.

33 Baris Tan and Andrea Matta. The digital twin synchronization problem: Framework, formulations, and analysis. *IISE Transactions*, 56(6):652–665, 2024. doi:10.1080/24725854.2023.2253869.

34 Louise Travé-Massuyès and Teresa Escobet. *BRIDGE: Matching Model-Based Diagnosis from FDI and DX Perspectives*, pages 153–175. Springer, June 2019. doi:10.1007/978-3-030-17728-7_7.

35 Tim Troendle. Simple simple: A simple building energy model. https://github.com/timtroendle/simple-simple, 2024. Accessed: 2024-09-04.

36 Lars B. Vosteen. FDIR for Digital Twins. Software, swhId: swh:1:dir:1972d8d2373f5d9c6e127fa16aec1a79755b056a (visited on 2024-11-14). URL: https://github.com/vosteen/FDIR.

37 Guanghui Zhou, Chao Zhang, Zhi Li, Kai Ding, and Chuang Wang. Knowledge-driven digital twin manufacturing cell towards intelligent manufacturing. *International Journal of Production Research*, 58(4):1034–1051, 2020. doi:10.1080/00207543.2019.1607978.