Contract-based Verification of Digital Twins

Muhammad Naeem and Cristina Seceleanu

School of Innovation, Design, and Engineering,
Mälardalen University, Västerås, Sweden
{muhammad.naeem, cristina.seceleanu}@mdu.se

Abstract. Digital twins are becoming powerful tools in industrial applications, offering virtual representations of cyber-physical systems. However, verification of these models remains a significant challenge due to the potentially large datasets used by the digital twin. This paper introduces an innovative methodology for verifying neural network-based digital twin models, in a black-box fashion, by integrating model checking into the process. The latter relies on defining and applying system-level contracts that capture the system's requirements, to verify the behavior of digital twin models, implemented in Simulink. We develop an automated solution that simulates the digital twin model for certain inputs, and feeds the predicted outputs together with the inputs to the contract model described as a network of timed automata in the Uppaal model checker. The latter verifies whether the predicted outputs fulfill the specified contracts. This approach allows us to identify scenarios where the digital twin's behavior fails to meet the contracts, without requiring the digital twin's design technicalities. We apply our method to a boiler system case study for which we identify prediction errors via contract verification. Our work demonstrates the effectiveness of integrating model checking with digital twin models for continuous improvement.

Keywords: Digital Twin · Model Checking · Uppaal· Neural Networks · Simulink

1 Introduction

The emergence of Industry 4.0 has propelled Digital Twins (DT) to the forefront of industrial digitalization, offering virtual representations of cyber-physical systems that facilitate precise simulations, analysis, and control [19]. As industrial systems grow in complexity, ensuring the reliability and correctness of these DT becomes crucial.

This work is a part of the Dynamic and Robust Distributed Systems (D-RODS) project aiming to develop an adaptable and dependable framework for the implementation and operation of DT, tackling issues in industrial digitalization including system integration, performance enhancement, and compatibility with legacy systems [8,17]. In the current study, we focus on the Verification and Validation (V&V) of Neural Network-based Digital Twins (NNDT) models using

model-checking techniques. NNDT have the potential to predict system behavior, conduct real-time tests, and identify optimization opportunities. However, their effectiveness depends on their ability to represent and accurately respond to real-world conditions [7].

Recent advancements in model-driven engineering, such as the CoCoSim framework, have demonstrated the value of integrating formal verification techniques with model-based design, especially for multi-periodic discrete systems [3].

Our research applies a similar concept to the domain of DT, proposing a novel approach that employs model checking to verify monotonicity, functional, and infrastructure contracts specified for black-box NNDT. We propose a methodology for systematically verifying the behavior of DT models implemented in `Simulink` by defining system contracts, as networks of Uppaal timed automata [1]. We present an automated solution that connects a simulation environment with model checking, allowing for the continuous validation and improvement of NNDT models. Our approach detects situations where the NNDT generates incorrect outcomes.

Due to their dynamic nature and complex interactions with cyber-physical systems, verifying DT models is challenging. Dahmen et al. emphasize the importance of systematic methods to verify and validate DT models [5], and propose an approach that breaks down the verification problem into smaller, more manageable components. This research aligns with our methodology of using (sub-)system contracts to verify specific aspects of DT behavior.

Given the importance of ensuring the correctness of DT behavior, our research aims to address the following research question: **RQ:** How can we design an automated solution to model check a black-box NNDT model?

To answer it, this paper presents a contract-based approach for verifying black-box NNDT models in Uppaal. The solution simulates the `Simulink` NNDT model for certain inputs, and feeds the predicted outputs together with the inputs to the contract model described as a network of timed automata in the Uppaal model checker. The latter verifies if the predicted outputs fulfill the specified contracts. In brief, our contributions are as follows:

- *Model-Checking-based Methodology:* We propose a methodology that uses contracts modeled as Uppaal timed automata, and employs model checking to verify the correctness of black-box NNDT.
- *System Contract Implementation:* We define a systematic approach for modeling system contracts that specify expected behaviors, enabling precise verification of model outputs against system requirements.
- *Practical Validation:* Our approach is illustrated through a burner-boiler system case study, showcasing its effectiveness on an industrial application.

Our approach facilitates the identification of scenarios where the DT's behavior fails to meet the contracts, without requiring knowledge of the model's internal structural and functional details.

The remainder of this paper is organized as follows. Section 2 provides the necessary background information. Section 3 details our methodology for verifying NNDT models using contract-based model checking. Section 4 describes our

case study of a burner-boiler system, including the DT model implemented in `Simulink`, as well as the system's contracts. Section 5 presents the implementation of contracts in UPPAAL, for model checking, and Section 6 discusses the verification results. Finally, Section 7 presents and compares to relevant related work, before Section 8 concludes the paper, summarizing our contributions and discussing potential lines of future work.

2 Preliminaries

This section presents the background information necessary to understand the concepts and methodologies employed in our study.

2.1 The *D-RODS* Approach

The *D-RODS* project, proposed a cutting-edge approach for development and operation of DT. This innovative framework aims to address the complexities inherent in industrial digitalization by integrating artificial intelligence (AI) and formal verification techniques.
Context: Physical (CP) represents the physical components including machineries and controllers.
Context: Learning (CL) layer (Fig.1a)) focuses on creating DT models through unsupervised learning. These models are derived from extensive data histories and existing domain expertise.

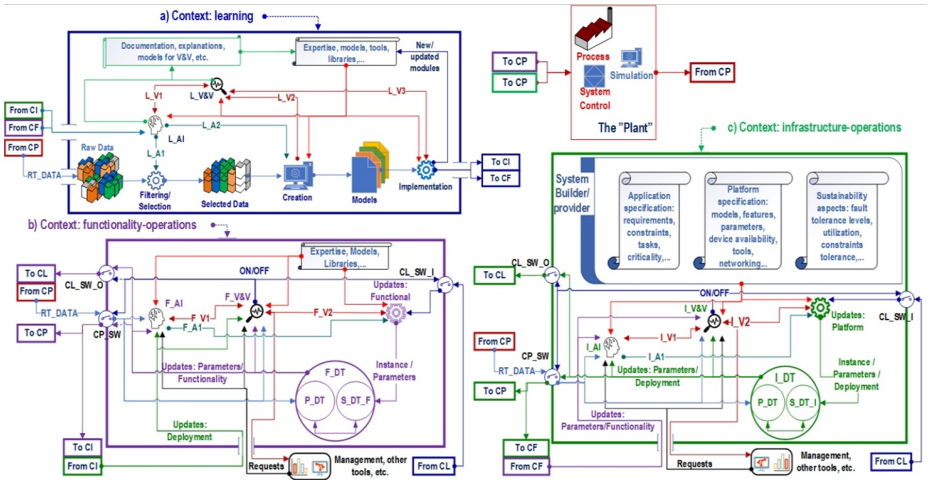


Fig. 1: D-RODS approach details

Context: Functionality (CF) and **Context: Infrastructure (CI)** are similar layers (Fig.1b, c), each containing AI and V&V components. These contexts

supervise and enhance the development and execution of complex DT, facilitating continuous learning, optimization, and behavior evaluation.

At its core, *D-RODS* seeks to unite DT, AI, and V&V technologies in a novel architectural setup. This integration aims to increase the trustworthiness of AI approaches through formal verification and analysis while optimizing operations, resource utilization, and power consumption. The framework is designed to support high levels of autonomy by ensuring the accuracy and efficiency of employed models through continuous learning and verification.

D-RODS is committed to improving system performance forecasting and optimizing resource allocation in the field of AI. The project also aims to integrate formal verification and runtime testing to ensure the accuracy of developed models and continuously monitor operational correctness. Through these advancements, *D-RODS* aims to advance DT technology, offering a comprehensive framework that addresses the challenges of complex industrial systems while promoting efficiency, reliability, and adaptability.

2.2 Uppaal Timed Automata

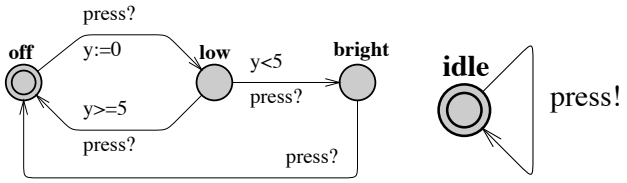
Model checking is a formal verification technique that systematically and exhaustively verifies whether a system model meets specified requirements. Uppaal [1] is a state-of-the-art model checker designed for modeling, verification, and simulation of distributed systems. Its modeling framework is based on Timed Automata (TA), which extends traditional finite-state machines by incorporating real-valued clocks. These clocks allow for the precise timing of events, enabling the modeling of systems where timing is critical.

In Uppaal, the concept of TA is further enhanced through the use of Uppaal Timed Automata (UTA). UTA introduces discrete data variables, such as integers and Boolean values, which can act as guards or expressions to control transitions between states. Additionally, UTA supports specifications in the C programming language, facilitating the observation and control of the model's discrete states.

A UTA can be formally defined as a tuple: $\langle L, l_0, C_h, V, E, I \rangle$, where: L is a finite set of locations in the automata model, l_o represents the initial location, $C_h = C_h! \cup C_h?$ denotes a set of channels for communication, with $C_h!$ and $C_h?$ representing sending and receiving channels, respectively [14]. V includes a set of data variables and clocks, E consists of edges connecting locations, I specifies invariants that must hold true for certain expressions.

Uppaal's ability to model and verify real-time systems makes it particularly valuable in safety-critical domains. The tool offers a user-friendly graphical interface for designing models and simulating their behavior, along with a command-line interface for efficient verification processes [15].

Fig 2(a) shows a *network of UTA* modeling a simple lamp and its user [16]. The lamp has three locations: off, low, and bright. If the user presses a button, i.e., synchronizes with the press? then the lamp is turned on. If the user presses the button again, the lamp is turned off. However, if the user is fast and rapidly presses the button twice, the lamp is turned on and becomes bright. The user



(a) Lamp (b) User

Fig. 2: The simple lamp example.

model is shown in Fig 2(b). The user can press the button randomly at any time or even not press the button at all. The clock y of the lamp is used to detect if the user is fast ($y < 5$) or slow ($y >= 5$).

The contracts (queries in UPPAAL) to be verified by model checking on the resulting network are specified in a decidable subset of (Timed) Computation Tree Logic ((T)CTL). In this paper, we verify (T)CTL queries of the following kinds (p is a state property): (i) Reachability: $E \Diamond p$ - the query evaluates to true if there exists a path where p eventually holds, and (ii) Invariance: $A \Box p$ - the query evaluates to true if (and only if) every reachable state satisfies p, in other words, for all paths p always holds.

3 Proposed Methodology

In this section, we describe our methodology for verifying NNDT models in a black-box fashion, by integrating model checking into the process.

3.1 Digital Twin Model as a Neural Network

The starting point of our methodology is a black box NNDT model, implemented in `Simulink` (in this paper). However, the approach can be applied to other modeling frameworks, straightforwardly. The NNDT serves as a virtual representation of the cyber-physical system, continuously updated with real-time data to mirror its current state and behavior.

The black-box nature of this neural network-based model allows us to capture complex, non-linear relationships within the system without requiring detailed knowledge of its internal structure. This approach provides flexibility and accuracy in representing the system's behavior across various operating conditions. The inputs and outputs form the basis for our subsequent contract verification process, allowing us to assess the NNDT's performance against predefined system contracts.

3.2 Contract Model Development

We develop a comprehensive contract model using UTA, and provide contract verification in Uppaal. These contracts capture the system's requirements and

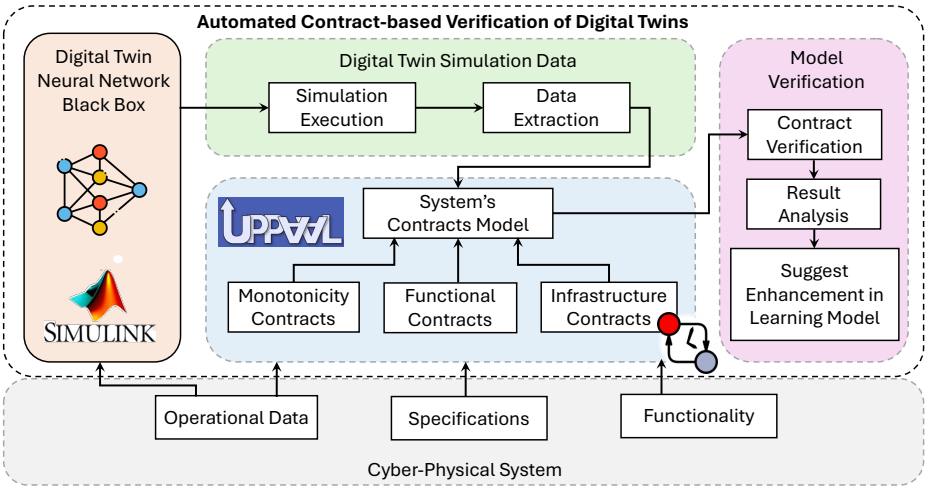


Fig. 3: Methodology

serve as a formal specification of the expected behavior of the NNDT. We categorize the contracts into three main types:

- *Monotonicity Contracts:* These contracts are designed to verify that the output of the NNDT behaves monotonically with respect to certain inputs. This means that if an input's value increases, the NNDT's output value should also increase, and vice versa. This property is crucial in many industrial applications where a predictable relationship between inputs and outputs is expected.
- *Functional Contracts:* These contracts specify the expected functional behavior of the digital twin, defining relationships between inputs and outputs that must be maintained throughout the operation of the model. These contracts ensure that the digital twin follows the intended operational specifications.
- *Infrastructure Contracts:* These contracts focus on the operational aspects of the digital twin, such as response times, resource usage, and system stability. These contracts ensure that the digital twin operates within acceptable performance limits.

Contracts for Neural Networks-based Digital Twins (NNDT). In the following, we define our notion of *NNDT contract*, as used in this paper.

Definition 1 (NNDT Contract). *Let $\mathcal{N} : \mathbb{R}^n \to \mathbb{R}^m$ be a digital twin represented by a neural network, mapping an input vector $x \in \mathbb{R}^n$ to an output vector $y \in \mathbb{R}^m$. We define the* assume-guarantee *contract for the digital twin, as the pair (A, G), as follows:*

- Assumption (A)*: We assume that the input and output of the digital twin at time t, denoted as $x(t)$, and $y(t)$, respectively, are represented by a moving*

average over the last m time steps (m is a constant that depends on the application), to mitigate the effects of inaccurate sensing, and predictions, respectively:

$$\hat{x}(t) = \frac{1}{m} \sum_{i=1}^{m} x(t-i), \tag{1}$$

where $\hat{x}(t)$ is the stabilized input, and:

$$\hat{y}(t) = \frac{1}{m} \sum_{i=1}^{m} y(t-i), \tag{2}$$

where $\hat{y}(t)$ is the stabilized output. To these cross-cutting assumptions (they hold for all contracts), we add predicate p that captures the requirements on inputs $x(t)$.

– Guarantee (G)*: A predicate q that captures the NNDT's expected output behavior. As an example, a* monotonicity contract *that states that, if the input $\hat{x}$ increases component-wise, then the output of the neural network, $\mathcal{N}(\hat{x})$, should also increase component-wise, can be expressed formally as follows (similarly, for the decreasing case):*

$$((\forall \hat{x}, \hat{x}' \in \mathbb{R}^n . \hat{x} \preceq \hat{x}'), (\mathcal{N}(\hat{x}) \preceq \mathcal{N}(\hat{x}')) \tag{3}$$

Here, $\preceq$ denotes component-wise inequality (that is, inequality that is applied individually to each entry, which can be the average value over m previous ones, as explained above). This ensures that the neural network maintains monotonicity and avoids erratic behavior in response to changes in input.

The semantics of the (A, G) contract of Definition 1 is given in terms of Uppaal *timed automata* (UTA), as follows.

Definition 2 (Semantics of NNDT Contracts). *A contract for a neural-network-based digital twin is a pair (A, G), where $A = ||_i UTA_{A_i}$ represents the network of $i \neq 0$* ***assumption UTA****, and $G = ||_j UTA_{G_j}$ is the network of $j \neq 0$* ***guarantee UTA****. The semantic contract is then defined by the parallel composition $A||G$.*

3.3 Role of Contracts in Verification

The contracts mentioned above play a critical role in the verification process because they set clear expectations for how the digital twin should behave. During the model-checking phase, we compare the outputs predicted by the NNDT model to these contracts. If the outputs do not meet any of the contracts, it shows that there might be a problem with how the model is working.

By using these contracts in a structured way, we can ensure that the digital twin meets its required functions and follows the necessary rules for operation. This helps to improve the reliability and safety of the system that it represents.

Verifying Satisfaction of NNDT Contracts. In our approach, the verification of NNDT contracts reduces to model checking the UTA parallel composition $A||G$ of Definition 2, against *invariance* properties of the form $A\Box\, q$, where q is the predicate that models the guarantee, per component. As sometimes it proves faster, we can also check *reachability* properties of the form $E <> \neg q$, where a witness trace returned by UPPAAL identifies a breach of contract.

3.4 Automated Verification Process

We develop an automated solution to bridge the gap between the `Simulink`-based NNDT and the UPPAAL contract model. This solution consists of the following steps:

- *Simulation Execution* The automated system triggers simulations of the DT model in `Simulink` across various input scenarios.
- *Data Extraction* The system captures the text output data generated by the `Simulink` model during these simulations.
- *Contract Verification* The captured data is fed into the UPPAAL contract model obtained as described above, which is then model checked against the specified contracts, hence verifying the NNDT's behavior compliance.

3.5 Results Analysis

We analyze the verification results, identifying any violations of the contracts, where the NNDT's behavior fails to meet the contracts. This process involves:

- Pinpointing the exact input conditions that lead to contract violations.
- Categorizing errors based on the type of contract breached (monotonicity, functional, or infrastructure).

4 Case Study: Burner-Boiler System

To illustrate our approach, we apply it on a case study focused on heating different liquids until they evaporate (Fig 4). This model includes several parts that work together to achieve the goal.

At the center of the system, there is a *wood provider*, which supplies fuel to a *warehouse*. From there, a *burner system* takes the fuel to heat up a *boiler*. The boiler heats containers made of various materials, each holding different types of liquids. The system's complexity comes from several randomly assigned rates. These include how efficiently the wood burns, how well heat transfers in the containers, and how quickly each liquid heats up. Key parameters, such as the starting amounts of wood and liquid, sizes of wood deliveries, and the evaporation temperatures of the liquids, are also randomly set. This randomness allows for generating a variety of scenarios in our study.

Fig 4 shows a simple diagram of this setup, highlighting the main parts, as well as how information flows through the system. The model is built around the *System of Interest (SoI)*, which includes three main subsystems:

– *Warehouse:* The central place for storing and distributing fuel.
– *Burner:* Responsible for burning fuel and generating heat.
– *Boiler:* The part where liquids are heated.

There is also an external *Wood Delivery System* that interacts with the SoI but operates separately. While this system is important for supplying wood, it is not included in our DT modeling. We can change its delivery conditions—like when and how much wood is delivered—but it remains outside our current focus.

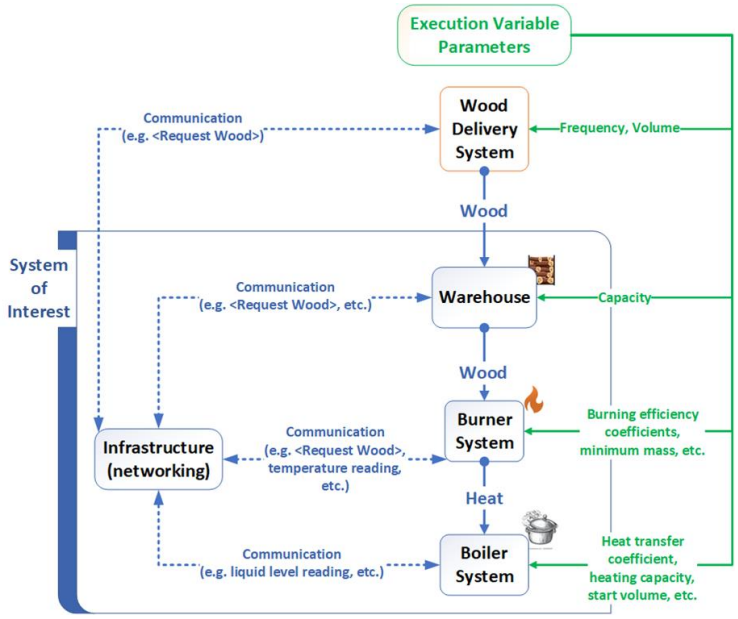


Fig. 4: The heating system.

Our main goal is to create efficient and accurate DT models. In this paper, we focus on the verification of a black-box NNDT model developed in our previous work. This model represents the complex interactions within the burner-boiler system, and our current efforts aim to ensure its correctness and reliability using *contract-based verification* in the Uppaal tool.

4.1 Digital Twin Model

The Fig 5 illustrates the DT model in `Simulink`. The NNDT model for the burner-boiler system is implemented in `Simulink` and serves as a virtual representation of the physical boiler. This model integrates real-time data to accurately reflect the current state and behavior of the boiler.

We use the following notation to describe the DT model for this use case:

B_T: Burner Temperature	Bo_T: Boiler Temperature
W_T: Water Temperature	W_M: Water Mass
Wo_M: Wood Mass	T_{env}: Environmental Temperature
t: Current time step	$T_{\mathbf{Boil}}$: Boiling Temperature
W_A: Water Alarm	Wo_D: Wood Delivered
Wo_A: Wood Alarm	Wo_R: Wood Request
A: Assumption	G: Guarantee

The model takes several input parameters, including temperatures, burner characteristics (Alpha, Beta, Delta), wood and water mass and alarms. These inputs allow the DT to adapt to changing conditions and predict the boiler's operations effectively.

Key outputs from the model include predictions for B_T, Bo_T, W_M, Wo_M, W_A, Wo_A, Wo_R and Wo_D. The DT uses a neural network to process these inputs and generate accurate predictions about future states of the system.

While the model has been trained on specific datasets and validated with known data, there remains a potential for errors due to limitations in the available data. This aspect is critical as it highlights the scope of our work in this paper. We focus on model-checking the NNDT black-box model using the contract-based approach described in Section 3.

Through the use of its neural network architecture, this DT captures complex relationships within the boiler system. Our contract-based verification method enables us to capture specific behavioral requirements for the NNDT, allowing us to assess its contract compliance. This approach ultimately aims to improve the reliability and accuracy of real-world applications NNDT.

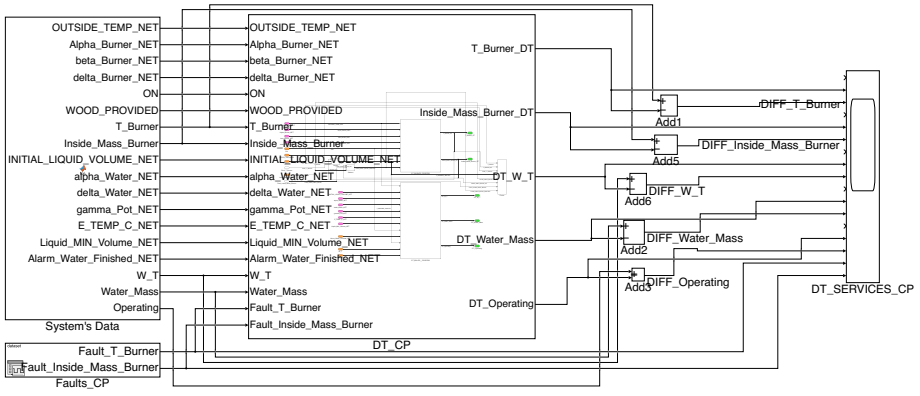


Fig. 5: DT Model

4.2 System Contracts

We define three categories of contracts for our DT model: *Monotonicity Contracts (MC)*, *Functionality Contracts (FC)*, and *Infrastructure Contracts (IC)*. Each contract is formulated as a pair, (A, G), using the notation described in Section 3.2. In cases where multiple assumptions exist, we have: $A = \wedge_i A_i$, and similar for multiple guarantees: $G = \wedge_i G_i$.

Monotonicity Contracts (MC) These contracts are designed to verify that the output of the NNDT behaves monotonically with respect to certain inputs, which is a crucial property of NN.

MC1 (B_T vs Bo_T): $A1: \forall t, \quad B_T(t) < T_{\text{Boil}}$

$$A2: \hat{B_T}(t) > \hat{B_T}(t-1) \ , \ G1: \hat{Bo_T}(t) \geq \hat{Bo_T}(t-1)$$
$$A3: \hat{B_T}(t) < \hat{B_T}(t-1) \ , \ G2: \hat{Bo_T}(t) \leq \hat{Bo_T}(t-1)$$

When the burner temperature $\hat{B_T}$ is below the boiling point T_{Boil}, it may increase or decrease smoothly over time. The boiler temperature $\hat{Bo_T}$ must follow the trend of the burner temperature, either increasing or decreasing, accordingly.

MC2 (W_M vs Bo_T):

$$A1: \hat{Bo_T}(t) > T_{\text{Boil}} \ , \ G1: \hat{W_M}(t) \leq \hat{W_M}(t-1)$$
$$A2: \hat{Bo_T}(t) \leq T_{\text{Boil}} \ , \ G2: \hat{W_M}(t) \approx \hat{W_M}(t-1)$$

When $\hat{Bo_T}$ exceeds T_{Boil}, evaporation occurs, causing W_M to decrease over time. However, if the $\hat{Bo_T}$ drops to or below T_{Boil}, the evaporation should slow down or stop, stabilizing the $\hat{W_M}$. This ensures that the system correctly models the evaporation dynamics, maintaining a realistic relationship between Bo_T and W_M.

MC3 (Wo_M vs B_T): $A1: \forall t, Burner = ON$

$$A2: \hat{Wo_M}(t) > \hat{Wo_M}(t-1) \ , \ G1: \hat{B_T}(t) \geq \hat{B_T}(t-1)$$
$$A3: \hat{Wo_M}(t) < \hat{Wo_M}(t-1) \ , \ G2: \hat{B_T}(t) \leq \hat{B_T}(t-1)$$

$\hat{B_T}$ depends on the $\hat{Wo_M}$. When wood burns, its mass decreases. If more wood is added, $\hat{B_T}$ should increase or stay the same. If wood mass decreases, $\hat{B_T}$ should eventually drop.

By establishing these monotonic relationships, the verification process ensures that the digital twin accurately simulates the boiler system's thermal behavior. Any contract violation suggests issues in the neural network model, data handling, or system design, helping in detecting potential problems.

Functionality Contracts (FC) The functionality contracts define the expected operational behavior of the system, by specifying how different components should respond to certain conditions.

1- Burner System FC:

The *burner system* ensures safe and efficient operation by managing wood supply, monitoring temperature, and responding to boiler signals.

FC1: $A:\ \hat{WO_M}(t) < WO_{M_{min}}\ ,\ G:$ RequestWood()
The burner must request wood when the $\hat{WO_M}$ falls below a certain threshold.

FC2: $A1:\ (WO_R(t) \wedge \neg WO_D(t + \Delta t)$ where $\Delta t = 60\text{s})\ ,\ A2:\ (\hat{WO_M} < WO_{M_{min}})\ , G:\ W_A$
The W_A should be triggered if no wood is received within 1 minute after the wood request, or when $\hat{WO_M}$ falls below the minimum level.

FC3: $A:\ \hat{WO_M}(t) > WO_{min}\ ,\ G1:\ \neg WO_R(t)\ ,\ G2:\ \neg WO_D(t)$
This states that there should be no wood request or delivery when the $\hat{WO_M}$ is above WO_{min}.

FC4: $A1:\ ReachedIdealRange(t) = \text{True}\ ,\ A2:\ \hat{B_T}(t) < 130°C \vee \hat{B_T}(t) > 160°C)\ ,\ G:$ CriticalTemperatureAlarm(t)
The system must enable the *CriticalTemperatureAlarm*, if $\hat{B_T}$ leaves the ideal operational range after reaching once.

FC5: $A:\ TurnOffSignal(t) = \text{True}\ ,\ G:$ Burner = Off
The burner must shut down immediately when it receives a *TurnOffSignal* from the boiler.

FC6: $A:\ \hat{W_M}(t) > W_{min}\ ,\ G:\ \hat{WO_M}(t) > 0$
The $\hat{WO_M}$ must always remain above zero if the $\hat{W_M}$ is above the minimum level, to keep the burner functional.

2- Boiler System FC:

The *boiler system* ensures stable operation by maintaining boiling conditions, monitoring water levels, and responding appropriately to critical conditions.

FC7: $A1:\ ReachedBoilingState(t)\ , A2:\ \hat{W_M}(t) > W_{M_{min}},\ G:\ BO_T(t) \approx T_{\text{Boil}}$
If the boiler system has reached the boiling state and the $\hat{W_M}$ is above the minimum level, the $\hat{BO_T}$ should remain around T_{Boil}.

FC8: $A:\ \hat{W_M}(t) < W_{M_{min}}\ ,\ G1:\ W_A(t)\ , G2:\ TurnOffSignal(t)$
If the $\hat{W_M}$ falls below the minimum level, the system must generate W_A and send a *TurnOffSignal* to the burner to prevent unsafe operation.

FC9: $A:\ \hat{W_M}(t) > W_{M_{min}}\ ,\ G1:\ \neg W_A(t)\ ,\ G2:\ \neg TurnOffSignal(t)$
When the $\hat{W_M}$ is above the minimum level, W_A should not be triggered, and no *TurnOffSignal* should be sent to the burner.

FC10: $A1:\ \forall t, Burner = off\ ,\ G1:\ \hat{B_T}(t) \geq T_{env}(t),\ ,\ G2:\ \hat{BO_T}(t) \geq T_{env}(t)$
The B_T and BO_T should always stay above or equal to the T_{env}, even if the burner is off. They may gradually decrease but should never drop below the ambient temperature, unless an external cooling source is applied.

4.3 Infrastructure Contracts (IC)

IC1: A : $\exists t, t_0$ such that (CriticalAlarm(t_0) = True) $\wedge$ ($\forall t' \in [t_0, t_0 + 5]$, CriticalAlarm($t'$) = True) , G : ShutDownBurner()
If a critical alarm remains active for more than 5*minutes*, the burner must shut down to prevent system damage and ensure safety.

5 Contract Modeling in Uppaal

The UTA models presented in this section demonstrate how contract-based verification can check the correctness of a digital twin model, systematically. By mapping component and system contracts onto Uppaal timed automata, we ensure that key constraints are validated. Uppaal allows for efficient verification, as it explores all possible states and detects contract violations.

5.1 Contract Model for MC1

The Monotonicity Contract (MC1) ensures that the Bo_T follows the changes in the B_T. This contract guarantees a realistic and predictable behavior of the heating system. The Uppaal model designed for MC1 is implemented as a timed automaton (Fig 6), effectively capturing the dynamic relationship between the temperatures. To model this contract in Uppaal, we used three templates: *UpdateV* (Variable Update), A_{MC1}, and G_{MC1}. The *UpdateV* template updates B_T and Bo_T values periodically. The A_{MC1} template represents the expected behavior of the B_T, while the G_{MC1} template ensures that the Bo_Tv follows the B_T's trend.

The *UpdateV* template maintains the system's temperature changes using global variables: B_T (current burner temperature), B_{T_1} (previous burner temperature), Bo_T (current boiler temperature), and Bo_{T_1} (previous boiler temperature). The function *UpdateVar()* updates these values using a moving average. A clock variable `(c)` ensures that updates occur at regular time steps.

A_{MC1} template defines three possible states for the burner temperature: `Increasing` ($B_T > B_{T_1}$), `Decreasing` ($B_T < B_{T_1}$), and `Stable` ($B_T == B_{T_1}$). The system transitions between these states based on the changes in B_T. If the burner reaches the boiling point, the system enters a special state (Boiling).

The G_{MC1} template enforces the expected reaction of the boiler temperature. It includes three states: `Increasing` ($Bo_T > Bo_{T_1}$), `Decreasing` ($Bo_T < Bo_{T_1}$), and `Stable` ($Bo_T == Bo_{T_1}$). The transitions ensure that the boiler temperature behaves consistently with the burner temperature.

Verification in Uppaal. Uppaal provides a query language to verify system properties [1]. The verification is done using queries, which check whether certain properties hold in all possible executions of the system. To verify the correctness of the model, we define the following queries:

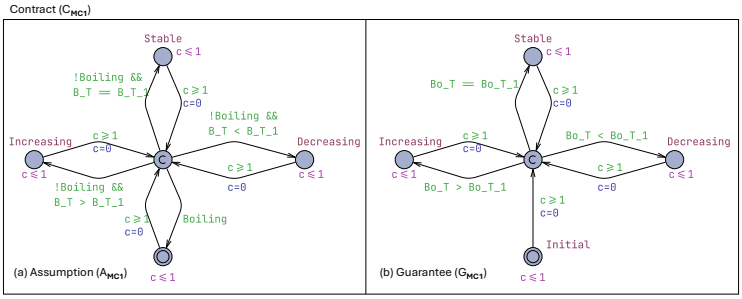


Fig. 6: Monotonicity Contract UTA Model for MC1

Safety Property Check: This ensures that the system always satisfies the monotonicity requirement:

```
A[] Assumption.Increasing imply Guarantee.Increasing
```

If this query fails, it indicates that there are case(s) where the boiler temperature does not always follow the burner temperature correctly. Additionally, UPPAAL provides diagnostic traces, which help locate errors by showing a counterexample when a property does not hold.

5.2 Contract Model for FC9

The Functional Contract (FC9) ensures that when W_M is above a defined minimum threshold, the alarm should not be triggered, and no turn-off signal should be sent to the burner. This contract prevents unnecessary interruptions in the heating system and ensures efficient operation.

In Fig 7, the UPPAAL model represents a contract (FC9) for a boiler system. To model this contract in UPPAAL, we define three key templates: A_{FC9} (Water Level Monitoring), $G1_{FC9}$ (Alarm Condition), and $G2_{FC9}$ (TurnOffSignal). Additionally, a UpdateV (Variable Update) template periodically updates the monitored water level, similerly as described in MC1 Model.

The A_{FC9} template maintains the state of the water level, defining two primary locations: `AboveW_min`, and `BelowW_min`, where . A clock variable (c) ensures regular updates, and the system transitions between these states based on the water level condition.

The $G1_{FC9}$ template models the alarm behavior by defining two states: `NotAlarm`, and `Alarm`. The transition to the Alarm state occurs only if the water level falls below the threshold. If the water level is above the threshold, the system remains in the NotAlarm state, ensuring that the contract requirement is met.

The $G2_{FC9}$ template models the burner control system by ensuring that the burner remains on as long as the water level is above the minimum threshold. The states in this template include `B_off_false`, and `B_off_true`. The transition

to `B_off_true` is allowed only if, ensuring that the burner is not turned off unnecessarily.

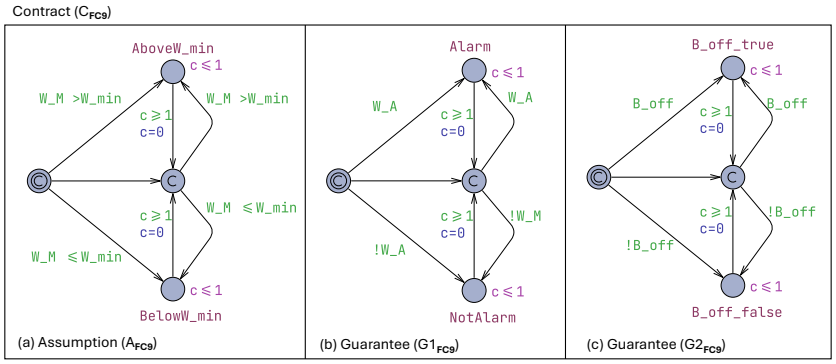


Fig. 7: Functionality Contract UTA Model for FC9

Verification in UPPAAL**.** To verify the correctness of the model, we define the following queries. The *invariance property check* ensures that, when the water level is above the threshold, neither the water alarm is triggered, nor the burner is turned off. This is expressed as:

```
A[] A_FC9.AboveW_min imply (not (G1_FC9.Alarm || G2_FC9.B_off_true))
```

If this property fails, it indicates that W_A or *turn-off* signal is incorrectly triggered despite the water level being sufficient. The reachability check verifies whether a violation of the contract is possible:

```
E<> A_FC9.AboveW_min imply (G1_FC9.Alarm || G2_FC9.B_off_true)
```

6 Verification Results

Our contract-based verification approach using UPPAAL revealed some scenarios in which the predictions of the NNDT model deviated from the specified contracts. These findings highlight areas for improvement in the model and potential safety concerns regarding the boiler system's operation.

6.1 MC1: Relation between the burner and the boiler temperatures

We simulate the contract automaton model described in section 5.1 to verify the relation between the burner and the boiler temperatures. The results illustrate a violation of the expected relationship between burner and boiler temperatures.

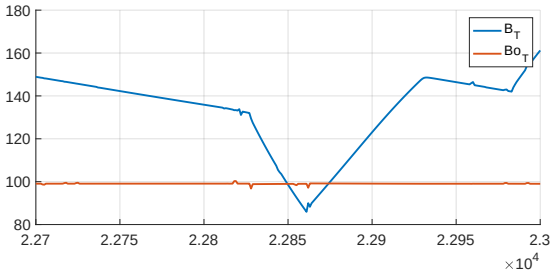


Fig. 8: Contract violation in MC1

According to MC1, the boiler temperature should correlate with the burner temperature, allowing for an acceptable delay δ, particularly when the burner temperature is below the boiling threshold.

In Figure 8, the graph shows that the burner temperature gradually decreases with time, even dropping below the boiling temperature at certain points. Despite this, the boiler temperature remains unchanged and does not follow the expected correlation. This discrepancy suggests that the heat transfer dynamics between the burner and the boiler are not accurately trained in the digital twin model.

The impact of this contract breach is significant. It may result in erroneous system behavior, such as the boiler maintaining a high temperature despite a decrease in burner temperature, which could lead to incorrect operational conditions. This conclusion emphasizes the need for model refinement to ensure that the digital twin accurately represents temperature dependencies and state transitions, enhancing its fidelity and reliability as a predictive tool.

6.2 FC3: Erroneous wood request

The results indicate a contract violation in the wood request mechanism, leading to an unnecessary wood supply. According to FC3, a wood request should only be triggered when the wood mass falls below the minimum threshold. However, as shown in Figure 9, noise in the wood request signal causes the system to falsely detect a low wood level, even when there is enough wood available.

As a result, the model requests unnecessarily additional wood, leading to an excessive amount of wood in the burner. This misbehavior can affect system efficiency and fuel management, potentially causing overheating or inefficient combustion.

Improvements in the wood request signal processing are required to avoid such errors. Possible solutions include noise-filtering techniques or threshold adjustments to avoid false triggers and ensure accurate wood requests.

These verification results highlight the importance of refining the DT model, particularly in areas of wood mass management, temperature control, and system state consistency. They also demonstrate the effectiveness of contract-based

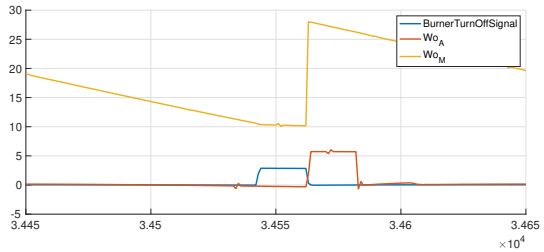


Fig. 9: Contract violation in FC3

verification in identifying potential safety and operational issues that might not be apparent through traditional testing methods.

6.3 FC9: Alarm Water Finished False Trigger

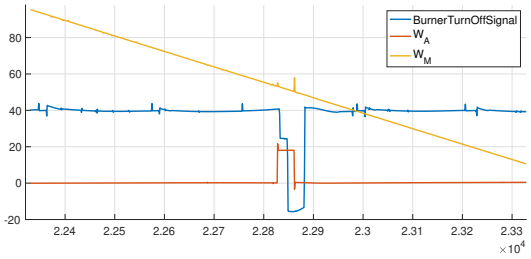


Fig. 10: Contract violation in FC9

The findings highlight a contract violation in the Water_Finished_Alarm, which led to an incorrect shutdown of the burner. According to FC9, the alarm should activate only when the water mass falls below the minimum threshold. However, as illustrated in figure 10, interference in the alarm signal leads to a false burner shutdown, even when the water mass remains above the minimum level.

Improvements in alarm signal processing are necessary to address this issue. Possible solutions include implementing signal filtering techniques, adjusting threshold conditions, or introducing hysteresis mechanisms to prevent minor fluctuations from causing false shutdowns.

7 Related Work

The verification of DT is a growing area of research due to its increasing use in critical applications.

Cimatti et al. provide the OCRA tool [4] that enables a *design-by-contract* methodology for reactive systems. In OCRA, the system's DT is its contract, which consists of an assumption on the system inputs, and a guarantee of the system outputs under the given assumptions, both described in Linear Temporal Logic. In comparison, our approach distinguishes between the DT, which is a neural network, and its contracts, and specifies the latter as timed automata that are fed with the DT's inputs and predicted outputs, to verify the black-box behavior of the DT by model checking.

Perhaps the closest research to ours is the approach used in CoCoSim [3], which is a toolbox that can be called directly from the Matlab Simulink environment, and can be used for code generation (e.g. Lustre) or property verification via contracts. The contracts are specified directly in Simulink, as model components, and the verification is carried out on the generated code, by using modular SMT-based verification engines (such as Zustre) that also generate assume-guarantee style formal contracts. However, the difference from our approach comes from the fact that the verification is not a black-box one, it instead uses the functional details of Simulink's model blocks that are transformed into code. The fact that our approach uses contracts modeled as simple timed automata that encode the properties to be model checked makes the verification very fast and scalable.

Notable work for *verifying neural networks*, which in our case describes the DT, has been carried out for a while now. Katz et al. introduce the "Reluplex" method [10], designed to verify deep neural networks by solving satisfiability problems, particularly in networks utilizing ReLU activation functions. In comparison, our method is agnostic of the NNDT's activation function, being based only on inputs, predicted outputs and their check of whether they satisfy the timed automata contracts.

Another strand of research focuses on *reachability analysis*, where methods such as *symbolic interval propagation* [11] and *abstract interpretation* [6, 12, 13, 18, 20] are used to estimate possible outputs of neural networks within a bounded input space. This is crucial for safety-critical systems to ensure that neural networks do not produce unexpected or unsafe outputs. For instance, the Neural Network Verification tool (NNV) [20] supports over-approximate analysis by combining the star set analysis used for feed-forward neural network controllers with zonotope-based analysis for nonlinear plant dynamics. In general, neural network verification approaches based on abstract interpretation are supported by tools that represent the inputs as a set, called abstract domain, which is then passed along the neural network, yielding a set that over-approximates the output that is used to evaluate the property under verification. The verification accuracy depends heavily on the abstract domain chosen [12]. If the abstraction is too rough, the tool might not be able to decide if the property is verified or falsified, therefore other methods need to be employed. By using contracts that encode the actual NNDT properties that need to be verified, we avoid such problems, providing a fast and scalable solution that can potentially be employed at run-time too, due to its insignificant overhead.

Additionally, researchers explore *probabilistic verification* [2] to handle the inherent uncertainty in data-driven models like neural networks. Techniques based on Bayesian methods and Monte Carlo simulations have been proposed to evaluate the performance and robustness of neural networks DT in dynamic environments. Kapteyn et al. [9] explore the mathematical foundations of digital twins, focusing on probabilistic graphical models, including dynamic Bayesian networks, to model complex systems and inform decision-making processes.

8 Conclusions and Future Work

This paper presents a novel methodology for verifying black-box neural network-based digital twin models using contract-based model checking. Our approach, which uses Uppaal timed automata to define system contracts, allows for black-box verification without requiring knowledge of the digital twin's internal details. This makes it particularly valuable for complex industrial applications. We validated our methodology using an existing neural network-based digital twin model of a boiler system developed in Simulink. By modeling contracts as Uppaal timed automata, we demonstrated how our approach can be applied to real-world industrial processes, ensuring digital twin predictions align with physical constraints and expected behaviors.

Our contract models show fast simulation times, making them suitable for runtime verification. This efficiency enables real-time monitoring of the digital twin's behavior during system operation without significantly impacting overall performance.

Future work will focus on analyzing the digital twin's performance under various input conditions and exploring ways to improve predictions for more complex systems. We aim to enhance the reliability and utility of digital twins in industrial applications, leading to improved efficiency and safety in operations.

Acknowledgments. The research was supported by vinnova's advanced digitalization program in the project D-RODS (ID: 2023-00244).

References

1. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal 4.0. Department of computer science, Aalborg university **1**(1), 1–48 (2006)
2. Boetius, D., Leue, S., Sutter, T.: Probabilistic verification of neural networks using branch and bound (2024), https://arxiv.org/abs/2405.17556
3. Bourbouh, H., Garoche, P.L., Loquen, T., Noulard, É., Pagetti, C.: CoCoSim, a code generation framework for control/command applications: An overview of CoCoSim for multi-periodic discrete Simulink models. Embedded Real Time Systems (ERTS) 2020 (ARC-E-DAA-TN74591) (2020)
4. Cimatti, A., Dorigatti, M., Tonetta, S.: Ocra: A tool for checking the refinement of temporal contracts. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 702–705 (2013)

5. Dahmen, U., Osterloh, T., Roßmann, J.: Verification and validation of digital twins and virtual testbeds. Int J Adv Appl Sci **11**(1), 47–64 (2022)
6. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE symposium on security and privacy (SP). p. 3–18. IEEE (2018)
7. Grieves, M., Vickers, J.: Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. Transdisciplinary perspectives on complex systems: New findings and approaches pp. 85–113 (2017)
8. Gu, R., Seceleanu, T., Xiong, N., Naeem, M.: A service-oriented digital twin framework for dynamic and robust distributed systems. In: 2024 IEEE International Conference on Software Services Engineering (SSE). pp. 66–73. IEEE (2024)
9. Kapteyn, M., Pretorius, J., Willcox, K.: A probabilistic graphical model foundation for enabling predictive digital twins at scale. Nat. Comput. Sci. **1(5)**, 337–347 (2021)
10. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In: 2017 Computer Aided Verification (CAV). vol. Lecture Notes in Computer Science, p. 97–117 (2017)
11. Kern, P., Büning, M.K., Sinz, C.: Optimized symbolic interval propagation for neural network verification (2022), https://arxiv.org/abs/2212.08567
12. Lemesle, A., Lehmann, J., Gall, T.L.: Neural network verification with pyrat (2024), https://arxiv.org/abs/2410.23903
13. Mazzucato, D., Urban, C.: Reduced Products of Abstract Domains for Fairness Certification of Neural Networks. In: Proc. of Static Analysis: 28th International Symposium, SAS 2021. p. 308–322. Springer-Verlag (2021)
14. Naeem, M.: Energy-efficient cyber-physical systems: A model-based approach (2024)
15. Naeem, M., Albano, M., Larsen, K.G., Nielsen, B., Høedholt, A., Laursen, C.Ø.: Battery aware analysis of sensor networks in uppaal smc. In: 2021 10th Mediterranean Conference on Embedded Computing (MECO). pp. 1–6. IEEE (2021)
16. Naeem, M., Albano, M., Larsen, K.G., Nielsen, B., Høedholt, A., Laursen, C.Ø.: Modelling and analysis of a sigfox-based iot network using uppaalsmc. IEEE Sensors Journal **23**(10), 10577–10587 (2023)
17. Seceleanu, T., Xiong, N., Enoiu, E.P., Seceleanu, C.: Building a digital twin framework for dynamic and robust distributed systems. In: International Conference on Engineering of Computer-Based Systems. pp. 254–258. Springer (2023)
18. Singh, G., Gehr, T., Püschel, M., Vechev, M.T.: An abstract domain for certifying neural networks. Proceedings of the ACM on Programming Languages **3**(POPL), 41:1–41:30 (2019)
19. Tao, F., Zhang, H., Liu, A., Nee, A.Y.: Digital twin in industry: State-of-the-art. IEEE Transactions on Industrial Informatics **15**(4), 2405–2415 (2018)
20. Tran, H.D., Yang, X., Lopez, D.M., Musau, P., Nguyen, L.V., Xiang, W., Bak, S., Johnson, T.T.: Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. Proc. of the 32nd International Conference on Computer-Aided Verification (CAV) (2020)