DTN-Core: Towards a Framework for Designing and Operating Digital Twin Networks

Majd Latah
Informatics Department
University of Hamburg
Hamburg, Germany
majd.latah@uni-hamburg.de

Mathias Fischer
Informatics Department
University of Hamburg
Hamburg, Germany
mathias.fischer@uni-hamburg.de

***Abstract*—This work presents, DTN-Core, a flexible and verifiable approach to create a network of digital twin models. The goal is to provide a generic design that supports location-independency and heterogeneity of messaging protocols, data formats, and data granularity. We also analyze data compatibility issues that may arise when coupling digital twins. A combination of data and behavioral coupling is used to represent each digital twin. A Timed automata approach is used as a real-time model that describes the behavior of the Digital Twin Network (DTN). System properties are defined using Timed Computation Tree Logic (TCTL) and the model is verified using an existing model-checking tool UPPAAL.**

***Index Terms*—Digital Twins, Digital Twin Networks**

I. Introduction

A Digital Twin (DT) is a virtual representation of an existing physical component [1]. A DT has to be updated by information exchanged between the physical and virtual components [2]. Digital twins can model complex systems and obtain conceptual information through visualization and simulation, where the difference between virtually generated and actual physical information can be compared and consequently used for better decision-making and operation enhancements [1]. DTs can transform simulations into digital replication that reflects the actual state of the physical component [1]. For example, a DT can be used to choose the optimal configuration that reduces the energy consumption in manufacturing systems through what-if simulations [3].

The Digital Twin Network (DTN) is a new concept introduced in [4], where multiple physical objects and digital twins can interact with each other using existing communication and data processing technologies. DTN, for example, can be used for modeling a complex system that requires collaboration among different existing entities [4]. For instance, DTN can be used to ensure resiliency and predict the future performance of 6G networks [5].

For digital twins, modularity is required to support interoperability and interchangeability [6]. In addition, DTs may not exist on a single node and should not be fully centralized [6]. Moreover, connected DTs are useful for cross-domain interactions when DTs belong to different domains and organizations [7]. For instance, when employed for optimizing the management of smart city services [8].

The main contribution of this paper is that we propose a generic and verifiable solution that allows DTs to be located at distributed locations. The framework allows the utilization of heterogeneous protocols with the ability to support different data formats and granularities at different levels of data and twin interactions (e.g., through functions and events).

II. Problem Statement

We consider the data coupling problem, which occurs when two different twins with similar or different levels of data granularity need to be connected. Due to the fact that digital twins may have different levels of granularity, integration issues may arise when these twins need to interact with each other. Therefore, the data compatibility among DTs must be verified beforehand.

Another problem is how we ensure that the proposed DTN meets the requirements of the system, which implies that the behavior of the DTN must be verified. To address this problem, we consider a behavioral coupling at different levels of twin interactions. We also focus on studying generic ways for messaging between DTs, as well as the establishment of a DTN based on existing standards.

The remainder of this paper is organized as follows. Section 3 discusses both requirements and related work. Section 4 presents our proposed framework (DTN-Core). Section 5 concludes the paper.

III. Requirements and Related Work

A. Requirements

To design a framework to create DTNs, we need to define the main requirements:

- Generic Design: The design should be applied to any DT type (i.e., application-agnostic).
- Heterogeneous Protocols: The system should allow DTs to utilize different messaging protocols and allow interactions with both digital and physical twins.
- Heterogeneous Data Formats: The system should allow DTs to utilize different data formats.
- Heterogeneous Granularity: the DTN should allow twins with different granularities to communicate with each other.

978-3-903176-66-9 ©2024 IFIP

- Location-independent: The system should allow local and remote DTs to be part of the DTN.
- Verifiable: the DTN model should be verified to ensure that it meets the requirements of the corresponding system.

B. Related Work

Related work is divided between concepts and modeling [7], [9], [10] and frameworks [11]–[15].

1) Concepts and Modeling: In [9], the authors introduced a conceptual design for a network of DTs, which is called the Digital Twin of a System (DTS). DTS represents a higher-level entity responsible for managing digital twins. The connection between DTS and digital twins is provided by a DTS2DT interface. The suggested prototype relies on the Asset Administration Shell (AAS) [16] for the digital representation of the digital twins and DTS.

A conceptual framework called Web of Digital Twins (WoDT) was introduced in [7]. The WoDT framework mainly targets cross-domain infrastructures and utilizes a semantic model based on distributed knowledge graphs.

In [10], the authors propose to create a DTN of an SDN-based network using knowledge graphs. The DTN manager parses application templates and utilizes knowledge graphs to create the DTN. The DTN application can request information and return the results obtained from the corresponding DT.

2) Frameworks: In [11], the authors presented CoTwin, a collaborative DT approach based on blockchain technology for improving DT models. The use case is a complex Cyber-Physical System (CPS) formed by highly connected cells of 5G and 6G networks. Smart contracts are used for storing trained models.

In [12], the authors proposed CPS Twinning, a framework for creating a virtualized network environment for testing and simulating DTs, which are automatically created from specifications. This work supports emulated and simulated DTs, where Mininet [17] is considered one of the core components of this work. The framework provides two modes of operation namely: replication and simulation. This work is designed mainly for security use cases.

In [13], the authors suggested an Application-driven Digital Twin Networking (ADTN) middleware for simplifying the interaction among DTs using IP-based protocols. Network resources are dynamically managed using Software-Defined Networking (SDN). This work utilizes Sensor Measurement Lists (SenML) as a unified data format.

In [14], the authors introduced OpenTwins, a framework for developing compositional digital twins, which are created from a combination of several digital twins. The system consists of an integration of different open-source tools such as Eclipse Ditto, Eclipse Hono, Grafana, InfluxDB, and Kafka-ML. While these tools are well-known, this design may inherit the limitations of the tools that are used to create the system.

Rail4Future [15] utilized FMI and SSP standards to create a platform for digital twins of a digitalized railway system. The authors used Jenkins pipelines along with a software repository to run the simulation process. One limitation of this approach is that it does not consider generic ways of supporting different protocols and data formats.

Unlike the previous frameworks, our framework relies on a generic approach that allows combining simulations and real systems regardless of communication protocols, messaging systems as well as data formats. We also ensure that the corresponding twins are compatible. Furthermore, the behavior of DTN is modeled and formally verified using model-checking approaches. Table I shows a comparison between related work and our proposed framework. From this table, it is clear that many approaches do not consider the heterogeneity of data formats and data granularity or the verifiability of the DTN.

TABLE I
A COMPARISON BETWEEN RELATED WORK AND OUR PROPOSED FRAMEWORK

	Generic Design	Heterogeneous Protocols	Heterogeneous Formats	Heterogeneous Granularity	Location-independent	Verifiable
DTS [9]	●	◑	◑	○	●	○
WoDT [7]	●	◑	◑	○	●	○
DTN for SDN [10]	◑	○	○	○	●	○
CoTwin [11]	◑	○	○	○	●	○
CPS Twinning [12]	◑	◑	○	○	○	○
ADTN [13]	◑	◑	○	○	●	○
OpenTwins [14]	●	◑	○	○	●	○
Rail4Future [15]	●	○	○	○	○	○
Our work	●	●	●	●	●	●

○ Not Considered ◑ Partially ● Fully Considered

IV. PROPOSED FRAMEWORK

Our framework, DTN-Core, relies on a generic messaging API along with a generic message parser/serializer. First, we need to describe source and destination twins. The digital twin that sends its outputs to another digital twin is called a source digital twin. Whereas a digital twin that receives its inputs from another digital twin is called a destination digital twin. For each source DT in the network, each DT has to check whether the corresponding destination DT is compatible or not. The DT also utilizes granularity transformers for connecting partially compatible twins. In DTN-Core, we propose a separation between execution and twin behaviors. A behavioral function can be performed before or after twin execution, and therefore such separation allows us to use the same execution file even when the behavior of the DT is changed. We define a behavioral model that describes the behavior of a given DT. The main system components are shown in Fig. 1, where each DT consists of the following components:

Generic Messaging API: We require a generic messaging API to allow DTs to interact with other DTs using different messaging systems. For example, a DT can receive messages using MQTT and then send the results using Modbus.

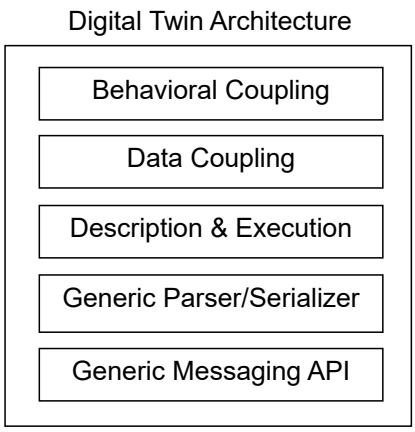


Fig. 1. Main components in DTN-Core

Generic Parser/Serializer: For handling data with different formats, we need a generic parser/serializer. The advantage is that we do not require DTs to use a unified format. Each message can be parsed and translated into different data types.

Description and Execution: A description file is used to describe the main function provided by the DT, the corresponding input/output variables, the behavioral model, and the corresponding interface along with the main supported protocols and data formats. In addition to the execution files that are needed to run the corresponding DT. A combination of Function Mockup Interface (FMI) [18] and System Structure and Parameterization (SSP) [19] standards can be used for this task, where SSP is used to provide the variable mapping.

Data Coupling: Each twin has a specific function that requires input data from another DT. In our design, each twin can interact with other DTs through functions and events. A function-based interaction requires verifying the variable mapping to ensure the data compatibility between the DTs. Therefore, a data coupling algorithm is necessary for checking the data compatibility of source and destination DTs. Creating granularity transformations within an adapter twin is needed for partially compatible twins (i.e., twins with different granularity levels) and also when interacting with other physical components such as Physical Twins (PTs).

Behavioral Coupling: A behavioral model is required for a representation of the behavior of a DT in DTN through functions and events. This is critical for real-time execution and representing the behavior of the DT. We choose the Timed Automata (TA) method [20] for the modeling of the behavior of DTs. To construct a DTN, we need to perform the following steps:

Step 1: Create description and execution files.
Step 2: Verify that the twins are data compatible.
Step 3: Create granularity levels and the adapter twin.
Step 4: Create and verify the behavioral model.

Note that the granularity levels and the adapter twin are not directly added to the DT since such information depends on the use case and cannot be directly inferred by the DT. The designer needs to add a file that contains a description of granularity levels as well as the corresponding adapter twin if the twins have different granularity levels.

Fig. 2. Granularity transformation using the adapter twin

A. Data Coupling of Digital Twins

To achieve a successful twin coupling, we need to determine whether the data of the corresponding twins are compatible or not. Both source and destination twins have their inputs and outputs described in the corresponding description and execution file. For each variable, an additional attribute is added to describe the granularity of different types of variables. Now we define two types of twin compatibility. Note that when the digital twin provides two levels of granularity of the same variable then it is considered a new variable in the corresponding description and execution file.

Definition 1: Two twins are fully data compatible if and only if the output variables of the source twin are from the same type and granularity as the ones of the destination twin

Definition 2: Two twins are partially data compatible if and only if the output variables of the source twin are from the same type as the destination twin and the granularity level of the source twin is higher than the granularity level of the destination twin.

Based on the previous definitions, two twins are incompatible in two cases. The first case is when at least one output variable of the source twin is from a different type than the input variables of the destination twin or when the granularity level of at least one output variable of the source twin has a lower granularity level than all input variables of the destination twin that are from the same type.

As an example of data coupling between two DTs, we consider DT_1 which performs network simulations and calculates future network utilization based on the current state, and DT_2 which aims to optimize the network utilization based on the results of DT_1. In this example, the output data of DT_1 is used as an input for DT_2, where DT_1 is assumed to be fully compatible with DT_2. However, a DT can be partially compatible with another DT/PT. In this case, an adapter-twin is added as a granularity transformer that allows the DT to interact with other DTs and their physical counterparts (see Fig. 2). Algorithm 1 is used to determine whether two twins are compatible or not. The algorithm verifies the variable mapping by checking the type and granularity level for each mapped variable pair. Fully compatible twins can be coupled directly without additional effort. Whereas partially

compatible twins require additional (high-to-low) granularity transformers. Algorithm 2 is used for coupling two DTs.

Algorithm 1: Verifying Variable Mapping (VVM)

Input: $Mapped\_Var\_Pairs$, GM;
/* GM is the granularity map */
/* T_Pairs is the set of transform (partially compatible) pairs */
/* G_Match is the number of variables with the same granularity level */
/* P_Compatible is the number of partially compatible variables */
/* S_Out is the set of output variables of the source twin */
T_Pairs = {∅, ∅};
$G\_Match = P\_Compatible = 0$;
$S\_Out = Out\_Var(Mapped\_Var\_Pairs)$;
foreach $v_i(src, dst) \in Mapped\_Var\_Pairs$ **do**
  if $Type(v_i(src)) == Type(v_i(dst))$ **then**
    if $Gran(GM, v_i) == Gran(GM, v_j)$ **then**
      $G\_Match++$;
    else if $Gran(GM, v_i) > Gran(GM, v_j)$ **then**
      $T\_Pairs = T\_Pairs \cup v_i(src, dst)$;
      $P\_Compatible++$;
    else
      continue;
  end
end
if $G\_Match == |S\_Out|$ **then**
  return $Fully\ Compatible$;
else if $G\_Match + P\_Compatible == |S\_Out|$ **then**
  return $Parially\ Compatible$, T_Pairs;
else
  return $Not\ Compatible$;

Algorithm 2: Data Coupling of Two DTs

Input: $Mapped\_Var\_Pairs$, GM;
/* GM is the granularity map */
/* D_In is the set of input variables of the destination twin */
$S\_Out = Out\_Var(Mapped\_Var\_Pairs)$;
$D\_In = In\_Var(Mapped\_Var\_Pairs)$;
$R, T\_Pairs = VVM(Mapped\_Var\_Pairs, GM)$;
if $R == Fully\ Compatible$ **then**
  foreach $Variable\ v_i \in S\_Out$ **do**
    $val = S\_Out(v_i)$;
    v_j = $Get\_Mapped\_Var(v_i)$;
    $D\_In(v_j) = S\_In(v_i)$;
  end
else if $R == Partially\ Compatible$ **then**
  $Adapter\_Twin(Mapped\_Var\_Pairs, GM)$;
else
  return $Not\ Compatible$;

Algorithm 2 is created on top of the Algorithm 1. First, we need to ensure the data compatibility through the previous algorithm. Then, we need to prepare the data for each mapped variable pair. Depending on the granularity level, the data can be transferred directly or it may need an additional granularity transformation based on the adapter twin (see Algorithm 3).

Algorithm 3: Adapter Twin

Input: $Mapped\_Var\_Pairs$, GM;
/* GM is the granularity map */
$S\_Out = Out\_Var(Mapped\_Var\_Pairs)$;
$D\_In = In\_Var(Mapped\_Var\_Pairs)$;
$R, T\_Pairs = VVM(Mapped\_Var\_Pairs, GM)$;
if $R\ != Not\ Compatible$ **then**
  foreach $Variable\ v_i \in S\_Out$ **do**
    if $v_i \in T\_Pairs$ **then**
      $val = S\_Out(v_i)$;
      v_j = $Get\_Mapped\_Var(v_i)$;
      $G\_v_i = Granularity(GM, v_i)$;
      $G\_v_j = Granularity(GM, v_j)$;
      $D\_In(v_j) = TF(val, G\_v_i, G\_v_j)$;
    else
      $D\_In(v_j) = S\_In(v_i)$;
    end
  end
else
  return $Not\ Compatible$;
end

B. Behavioral Coupling of the DTN

Modeling is useful for representing the behavior of DTN and also for verifying the requirements of the system. To model the DTN behavior, we consider Timed Automata (TA) [20], which includes time-state transitions with time constraints based on finite automata combined with a finite set of real-valued clocks [20]. This allows the system to be verified based on real-time requirements [20]. Each DT is represented using a separate TA, which is manually designed according to the intended behavior of each DT (see Fig. 3).

TA Transitions: represent twin interactions using functions and exchanged events. Each transition has a specific type. The internal type represents an interaction with an internal entity (e.g., a local DT or PT). The external type represents an interaction with an external entity (e.g., an external DT or PT). Additionally, each transition has a specific level (e.g., low-level and high-level). Each external transition has a specific entity which can be assigned as DT or PT. Each entity is also associated with a specific identifier (ID). In addition, each external transition is associated with a specific role. For example, a DT that provides input data to another remote DT can have a source (Src) role, whereas a DT that receives data from another remote DT can have a destination (Dst) role. For simplicity, when the twin has a destination role, we color the transition that represents the corresponding external transition with red color.

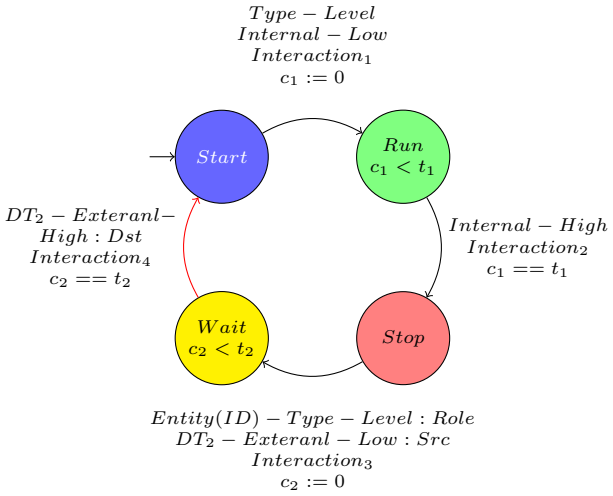


Fig. 3. Representing a DT behavior using TA

TA Nodes: represent a state of the DT, where each node has a specific type and level, which are determined based on the type and level of the last transition leading to that node.

Definition 3: A node type is external (internal) if and only if the type of the last transition leading to that node is external (internal).

Definition 4: A node level is low (high) if and only if the level of the last transition leading to that node is low (high).

Adding the colors to the TA nodes allows us to represent the DT without explicitly indicating the type of transition. The red ones are internal high-level nodes. The green ones are internal low-level nodes. The blue ones are external high-level nodes. The yellow ones are external low-level nodes.

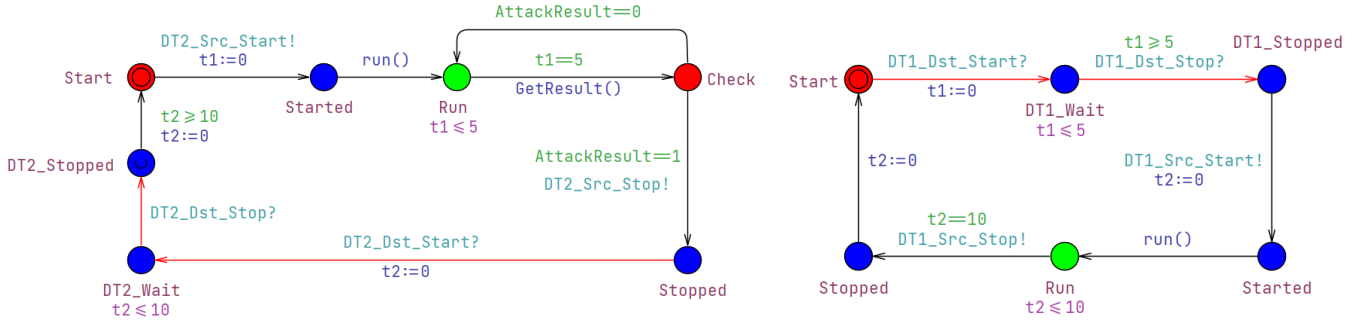
Algorithm 4 shows our proposed behavioral coupling model. The internal actions are applied directly to the local DT, whereas the external transitions require interaction with a remote DT based on a specific role. Therefore, the external transitions need to be associated with a remote DT as well as the corresponding role. As an example, we consider a simple DTN that consists of two DTs, where DT_1 and DT_2 spend 5 and 10 seconds, respectively. DT_2 depends on the output of DT_1. Fig. 4 shows how we can represent the DTN using

Algorithm 4: Behavioral Coupling of Two DTs

Input: LDT, RDT, TA, TTS
/* LDT: Local Digital Twin */
/* RDT: Remote Digital Twin */
/* TA: Timed Automata */
/* TTS: Total Time Steps */
/* The Clock is Simulated */
$t = 0$; /* Clock */
$TA.Do\_Step(X, 0)$; /* Start TA */
while $True$ **do**
  $t+ = 1$, $Res = X$;
  $S = TA.Get\_Current\_Stage()$;
  if S is $NewTrans$ **then**
    $T = S.Get\_Trans\_Type()$;
    $L = S.Get\_Trans\_Level()$;
    $C = S.Get\_Trans\_Clock()$;
    if $S.Type == External$ **then**
      $R = S.Get\_Twin\_Role()$;
      $Res = Apply(RDT, R, T, L, S.Trans)$;
    else
      $Res = Apply(LDT, T, L, S.Trans)$;
    end
    if $C == 0$ **then** $t = 0$;
  end
  if $t == TTS$ **then break**;
  $TA.Do\_Step(Res, t)$;
end

TA in UPPAAL [21], which is a well-known model-checking tool developed jointly by Uppsala University and Aalborg University for analyzing real-time systems through modeling, simulations as well as verification [21].

Fig. 4(a) represents the TA of DT_1, which we assume that it runs an anomaly detection model. The external high-level transition $DT_2\_Src\_Start$ represents a new event sent to DT_2, where the role of DT_1 in this transition is assigned as source. On the other hand, the internal low-level transition $run()$ represents calling the anomaly detection function of DT_1. DT_1 returns to the Run state as long as no attack is detected. DT_1



(a) TA of DT_1

(b) TA of DT_2

Fig. 4. Representing the DTN using TA in UPPAAL

needs to interact with DT_2 for further attack mitigation when a new attack is detected using $DT_2\_Src\_Stop$ transition.

Fig. 4(b) represents the TA of DT_2, which we assume that it runs an attack mitigation model. DT_2 remains in the DT_1_Wait state until receiving a new event from DT_1 using $DT_1\_Dst\_Stop$ transition, which indicates that a new attack is detected. DT_2 will send a new event to DT_1 using $DT_1\_Src\_Start$ transition. The next transition $run()$ represents calling the attack mitigation function of DT_2. Then, DT_2 will be in the Run state for 10 seconds. Then, it will inform DT_1 about the results using $DT_1\_Src\_Stop$ transition.

C. Defining System Properties

System properties φ are defined using Timed Computation Tree Logic (TCTL) [22], an extension of CTL [23]. TCTL allows quantitative temporal operators and therefore can be used for analyzing real-time systems [22]. TCTL formulas have the following grammar [24]:

$$\Phi ::= p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid E\ \Phi \mid A\ \Phi$$

where p is a proposition $\in AP$, a finite set of atomic propositions. An atomic proposition is a minimal declarative statement with a truth condition [25]. A path formula is defined using the following grammar [24]:

$$\varphi ::= \Phi_1 \cup_{\sim c} \Phi_2$$

where c $\in \mathbb{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$, U: until.

Common abbreviations [22]: $E \Diamond_{\sim c}$ for E (true $U_{\sim c}$ Φ), $A \Diamond_{\sim c}$ Φ for A (true $U_{\sim c}$ Φ), $E \Box_{\sim c}$ Φ for $\neg A \Diamond_{\sim c} \neg\Phi$, $A \Box_{\sim c}$ Φ for $\neg E \Diamond_{\sim c} \neg\Phi$. For instance, in our DTN when DT_2 is in the Run state, DT_1 has to wait 10 seconds. This property can be expressed in TCTL as follows [26]:

$$DT_2.Run \implies A \Diamond_{\leq 10}\ DT_1.Wait$$

D. Verifying the DTN Model

It is important to verify whether the designed DTN meets the system's requirements. Otherwise, we must provide a counter-example to show that the DTN model is faulty or unsafe. Model-checking [27] can be performed which allows us to check whether the TA of a given DTN satisfies the system properties defined using TCTL formulas [22]. This can be performed using existing tools such as UPPAAL [21]. We use UPPAAL query language to verify that DTN is deadlock-free in addition to verifying: reachability (e.g., activation of DT_1), safety (e.g., DT_1 and DT_2 do not run at the same time), and liveness (e.g., when DT_1 is running, eventually it will stop.

In Table II, for each category of system properties, we write a corresponding query (or a set of queries) that can be verified by UPPAAL. For instance, the query Q_1 is used to check whether the DTN is deadlock-free. The query Q_2 verifies whether the state Run is reachable. This is useful for sanity checks of the corresponding DT [28]. The query Q_3 ensures the safety of DTN by ensuring that only one DT is in the Run state at the same time. The query Q_4 verifies that whenever DT_1 is in the Run state, then DT_2 will be in the DT_1_Wait state. Similarly, the query Q_5 verifies that whenever DT_2 is in the Run state, then DT_1 will be in the DT_2_Wait state. The query Q_6 verifies that being in the Run state of DT_2 implies that DT_1 is in the DT_2_Wait state and $DT_1.t_2 \leq$10.

TABLE II
SYSTEM PROPERTIES AND CORRESPONDING QUERIES

	Property	**Query**
Q_1	Deadlock-free	A □ not deadlock
Q_2	Reachability	E $\Diamond$ DT_1.Run
Q_3	Safety	A □ not (DT_1.Run and DT_2.Run)
Q_4	Liveness	DT_1.Run $\implies$ $DT_2.DT_1$_Wait
Q_5	Liveness	DT_2.Run $\implies$ $DT_1.DT_2$_Wait
Q_6	Bounded Liveness	A □ (DT_2.Run imply $DT_1.DT_2$_Wait and $DT_1.t_2 \leq$10)

V. CONCLUSION

This work focused on proposing a generic and verifiable approach for creating digital twin networks. DTN-Core used two levels to describe the data and the behavioral model of each DT. The system allows local and remote digital twins to run simultaneously provided that the twins are compatible. Data compatibility is checked by verifying the variable mapping based on the type and granularity of the corresponding mapped pairs. Granularity transformers are introduced to allow partially compatible twins to interact with other DTs. TA is used to describe the behavioral model of each DT. The behavioral model of DTN is verified using UPPAAL according to predefined system properties.

REFERENCES

[1] M. Grieves, "Digital twin: manufacturing excellence through virtual factory replication," *White paper*, vol. 1, no. 2014, pp. 1–7, 2014.

[2] E. VanDerHorn and S. Mahadevan, "Digital twin: Generalization, characterization and implementation," *Decision support systems*, vol. 145, p. 113524, 2021.

[3] F. Pires, B. Ahmad, A. P. Moreira, and P. Leitão, "Digital twin based what-if simulation for energy management," in *2021 4th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*. IEEE, 2021, pp. 309–314.

[4] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 789–13 804, 2021.

[5] X. Lin, L. Kundu, C. Dick, E. Obiodu, T. Mostak, and M. Flaxman, "6g digital twin networks: From theory to practice," *IEEE Communications Magazine*, vol. 61, no. 11, pp. 72–78, 2023.

[6] H. Ahmadi, A. Nag, Z. Khar, K. Sayrafian, and S. Rahardja, "Networked twins and twins of networks: An overview on the relationship between digital twins and 6g," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 154–160, 2021.

[7] A. Ricci, A. Croatti, S. Mariani, S. Montagna, and M. Picone, "Web of digital twins," *ACM Transactions on Internet Technology*, vol. 22, no. 4, pp. 1–30, 2022.

[8] G. del Campo, E. Saavedra, L. Piovano, F. Luque, and A. Santamaria, "Virtual reality and internet of things based digital twin for smart city cross-domain interoperability," *Applied Sciences*, vol. 14, no. 7, p. 2747, 2024.

[9] L.-T. Reiche, C. S. Gundlach, G. F. Mewes, and A. Fay, "The digital twin of a system: A structure for networks of digital twins," in *2021 26th IEEE international conference on emerging technologies and factory automation (ETFA)*. IEEE, 2021, pp. 1–8.

[10] D. R. R. Raj, T. A. Shaik, A. Hirwe, P. Tammana, and K. Kataoka, "Building a digital twin network of sdn using knowledge graphs," *IEEE Access*, vol. 11, pp. 63 092–63 106, 2023.

[11] M. García-Valls and A. M. Chirivella-Ciruelos, "Cotwin: Collaborative improvement of digital twins enabled by blockchain," *Future Generation Computer Systems*, vol. 157, pp. 408–421, 2024.

[12] M. Eckhart and A. Ekelhart, "Towards security-aware virtual environments for digital twins," in *Proceedings of the 4th ACM workshop on cyber-physical system security*, 2018, pp. 61–72.
[13] P. Bellavista, C. Giannelli, M. Mamei, M. Mendula, and M. Picone, "Application-driven network-aware digital twin management in industrial edge environments," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 11, pp. 7791–7801, 2021.
[14] J. Robles, C. Martín, and M. Díaz, "Opentwins: An open-source framework for the development of next-gen compositional digital twins," *Computers in Industry*, vol. 152, p. 104007, 2023.
[15] O. Kugu, S. Zhou, R. Nowak, G. Müller, S. H. Reiterer, A. Meierhofer, S. Lachinger, L. Wurth, and M. Grafinger, "An fmi-and ssp-based model integration methodology for a digital twin platform of a holistic railway infrastructure system," in *Modelica Conferences*, 2023, pp. 717–726.
[16] "Details of the Asset Administration Shell - Part 1," https://www.plattform-i40.de/IP/Redaktion/EN/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V2.html, [Online accessed 03-July-2024].
[17] "Mininet," https://mininet.org/, [Online accessed 31-May-2024].
[18] "FMPy," https://fmpy.readthedocs.io/en/latest/, [Online accessed 31-May-2024].
[19] "SSP," https://ssp-standard.org/, [Online accessed 31-May-2024].
[20] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical computer science*, vol. 126, no. 2, pp. 183–235, 1994.
[21] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International journal on software tools for technology transfer*, vol. 1, pp. 134–152, 1997.
[22] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and computation*, vol. 104, no. 1, pp. 2–34, 1993.
[23] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Science of Computer programming*, vol. 2, no. 3, pp. 241–266, 1982.
[24] E. Khamespanah, R. Khosravi, and M. Sirjani, "An efficient tctl model checking algorithm and a reduction technique for verification of timed actor models," *Science of Computer Programming*, vol. 153, pp. 1–29, 2018.
[25] E. Akhmatova and D. Molla, "Recognizing textual entailment via atomic propositions," in *Machine Learning Challenges Workshop*. Springer, 2005, pp. 385–403.
[26] S. Yovine, "Kronos: A verification tool for real-time systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 1, no. 1-2, pp. 123–133, 1997.
[27] E. M. Clarke, "Model checking," in *Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17*. Springer, 1997, pp. 54–56.
[28] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," *Formal methods for the design of real-time systems*, pp. 200–236, 2004.