Twinning-by-Construction: Ensuring Correctness for Self-Adaptive Digital Twins

Eduard Kamburjan[1], Crystal Chang Din[2], Rudolf Schlatte[1],
S. Lizeth Tapia Tarifa[1], and Einar Broch Johnsen[1]

[1] University of Oslo, Oslo, Norway
{eduard,rudi,sltarifa,einarj}@ifi.uio.no
[2] University of Bergen, Bergen, Norway
crystal.din@uib.no

Abstract. Digital twin applications use digital artefacts to twin physical systems. The purpose is to continuously mirror the structure and behavior of the physical system, such that users can analyse the physical system by means of the digital twin. However, the physical system might change over time. In this case, the digital twin's ensemble of digital artefacts needs to be reconfigured to correctly twin the physical system again. This paper considers a digital twin infrastructure combining MAPE-K feedback loops and semantic reflection to automatically ensure that the digital artefacts correctly twin the physical system; i.e., the resulting system is *twinned-by-construction*. We consider the monitoring of both structural and temporal correctness properties for digital twin, including the time delay required by reconfiguration, and the capture of execution traces to reflect digital threads in the digital twin framework.

1 Introduction

Digital twins are a major innovation driver for the digitisation of key industries. Digital twin applications use digital artefacts to continuously mirror a physical system, such that users can analyse the physical system by means of the digital twin. At their core, they describe applications where a physical system, the *physical twin (PT)*, is mirrored structurally and behaviourally by some digital system, the *digital twin (DT)*; this mirroring turns the DT into a live replica of the PT. It is crucial that the PT and the DT interact with each other; i.e., data flows between them in both directions such that the DT can detect changes in the PT and perform actions. The physical system might change over time. In this case, the digital twin's ensemble of digital artefacts needs to be reconfigured to correctly twin the physical system again. Over its liftetime, the DT can then be seen as a trace of different configurations of digital artefacts, reflecting the changes to the PT. In this paper, we consider DTs that explicitly mirror changes to the PT in the digital system, enabling the user to access the *digital thread* of a physical asset.

In this paper, we discuss the reconfiguration of digital twins from the perspective of X-by-construction (XbC) approaches and self-adaptive systems, and

how these reconfigurations can be monitored. We say that a physical system is *twinned* by a digital system if the digital system has the same structure and behavior as the physical system. Static digital twins, i.e., twins that mirror a structure that does not change over time, are *twinned-by-construction* if the initialisation of the digital system ensures this twinning property. However, when the PT evolves over time, self-adaptation at runtime will play a crucial role in not only keeping the DT running, but also to re-establish its twinning property.

To illustrate these interactions between a self-adaptive DT and the PT, we consider an *architecture* for digital twins based on MAPE-K feedback loops [1,2,3] (Monitor, Analyse, Plan, and Execute based on Knowledge), a well-known method to organize autonomous systems. We distinguish between two different MAPE-K feedback loops for the *digital twin units* and the *digital twin infrastructure* in a digital twin application, where a DT unit is a digital artefact, for example, a simulator that mirrors exactly one part of the physical system, and the DT infrastructure orchestrates an ensemble of DT units.

One of the MAPE-K feedback loops uses the DT infrastructure to make sure that the DT units indeed jointly mirror the PT. Twinning-by-construction (TbC) ensures that the DT infrastructure connects the DT units in such a way that the DT mirrors the structure of the PT. If the physical system changes over time, the DT infrastructure needs to adapt and *re-twin* the digital system and re-establish the structural and behavioural correspondence between PT and DT. The DT infrastructure also realizes other features of the application, e.g., the above-mentioned data flow, the user interfaces, and analysis support over the DT such as the exploration of speculative scenarios, which again can be realised via a second MAPE-K feedback loop that only focuses on the behavioural correspondence between PT and DT.

This paper mainly focuses on the structural correspondence between the PT and the DT, as shown in Fig. 1, and discusses the connection between the concept of twinned-by-construction and self-adaptation at runtime by combining MAPE-K feedback loops with semantically lifted programs [4], and runtime monitoring of properties for self-adaptive systems. Semantically lifted programs combine knowledge graphs and object-oriented programming languages by enabling the program to *semantically reflect* on itself as a knowledge graph. For digital twin applications, we implement the DT units by means of functional mock-up units (FMUs) [5,6] and use ontology-based *asset models* to connect the state of the DT and the PT [7] (see Fig.1). Knowledge graphs enable a uniform treatment of the DT infrastructure (via semantic reflection) and a physical system (via the asset model). Users can send queries to both the asset model and the DT.

Furthermore, we consider *trace-based TbC* for digital twin applications. Correct twinning captures the property that the current structure of the DT corresponds to the current structure of the PT, as it is expressed in the asset model. Trace-based TbC strengthens this notion of correctness by requiring that the execution *trace* of the DT must also mirror the (execution) *trace* of the PT. Based on this constraint, information concerning physical twin units that are no longer present in the *current* configuration of the PT, but were previously
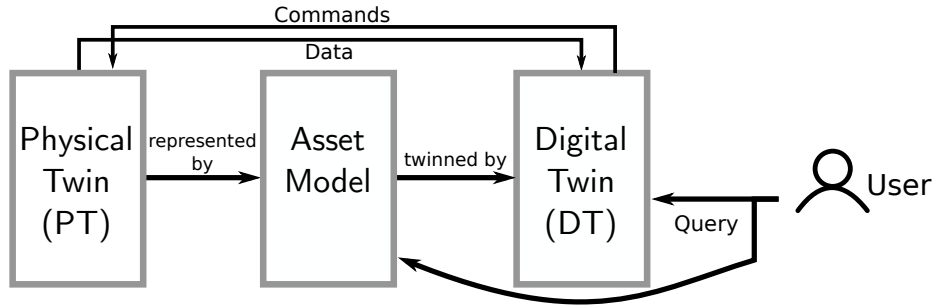
Fig. 1. High-level description of a digital twin architecture: the asset model serves as an interface for structural changes between digital twin (DT) and physical system (PS). Knowledge graphs enable uniform access to both DT and asset model.

present, can be found in the trace of the DT. Consequently, information can be retrieved about DT units corresponding to PT units that no longer exist. Trace-based TbC connects digital twins to the concept of digital threads (see, e.g., [8,9]), which has not been explored so far for DTs using ontologies and asset models.

Contributions and Structure. Our main contributions are (1) a detailed discussion that establishes a conceptual link between digital twins, X-by-construction and self-adaptation at runtime, and (2) an extension of semantical lifting that connects the digital thread with asset models. We introduce the notion of *twinned-by-construction* in Sec. 2 and the role of self-adaptation at runtime in Sec. 3. Sec. 4 broadens the discussion to digital threads for semantically lifted programs. We discuss related work and conclude the paper in Sec. 5 and Sec. 6

2 Twinned-by-Construction Systems

In this section we discuss twinned-by-construction (TbC) for static digital twins, and the role played by knowledge graphs. We introduce the basics by example and refrain from giving a formal introduction of the technology stack. For a summary of the use of knowledge graphs in digital twins and asset models, we refer to [7]; for a general introduction, we refer to [10].

Asset Model. An asset model serves as an interface between a physical and a digital system. It is a database (or a file) containing an organized description of the composition and properties of assets. An asset model is useful in a digital twin context because it can provide the twin with static configuration data for the digital twin units. The DT infrastructure is responsible for the updates and synchronization between the asset model and the DT. We assume here that the asset model is expressed as, or can be converted into, a knowledge graph.

Example 1. Consider a digital twin that, for some building construction, is twinning the structure of the walls. The building has two walls at the moment: one is left of the other one. As a knowledge graph, this is expressed using the following triples in RDF [11]:

```
as:InProd rdfs:subClassOf as:Asset.
as:Wall   rdfs:subClassOf as:Asset.
as:w1 a as:Wall.       as:w2 a as:Wall.
as:w1 a as:InProd.     as:w2 a as:InProd.
as:w1 as:leftOf as:w2.
```

Each RDF triple consists of three nodes, where in the example each node is either a name of the form `prefix:name`, or the symbol `a` expressing that the first node belongs to the third node, which must be a class. In the example, the first two lines define three classes, the class of all assets (`as:Asset`), and its subclasses of all walls (`as:Wall`) and all assets which are actively used in production `as:InProd`. Lines 3-4 define two objects `as:w1` and `as:w2` which are walls used in production. The last line defined that `as:w1` is left of `as:w2`. The order between the triples listed in the asset model does not matter.

Twinned-by-Construction. If the physical system does not change, the DT can be statically checked to ensure that it correctly mirrors the structure of the PT by statically comparing the asset model with the structure described in the DT infrastructure. Since the core principle of the digital twin is the structural mirroring, the asset model can be used as a guideline to construct the digital twin. In this case, we say that the digital twin is *twinned-by-construction* and the task of establishing the structural connection between PT and DT using an asset model is *twinning-by-construction*.

Figure 2 shows the structure of a correctly twinned system with two physical twin units (e.g., two walls `Asset 1` and `Asset 2`), their two descriptions in the asset model (e.g., `w1` and `w2` in the RDF given above) and two digital twin units `DTU 1` and `DTU 2` (e.g., two wall simulators): Any DT unit twins some part of the asset model, which in turn represents a PT unit. The DT unit and the PT unit are directly connected by data flowing between them. The correct twinning is established over *all* DT and PT units, as the interconnections in the physical system must also be mirrored in the digital one. For Example 1, this would be that the wall simulators are connected according to the `asset:leftOf` guideline.

Given a fixed asset model, an existing DT infrastructure, and a mapping from asset classes to DT units, one can easily define the corresponding structure in the digital twin: first, for each class of the asset define a class in the application with the same spatial information and the used simulation unit. Second, for each object in the asset model create an object of the corresponding class, and replicate the spatial information. We elide the details of this construction, which is the current way to write digital twins, but point out that such systems are already TbC.
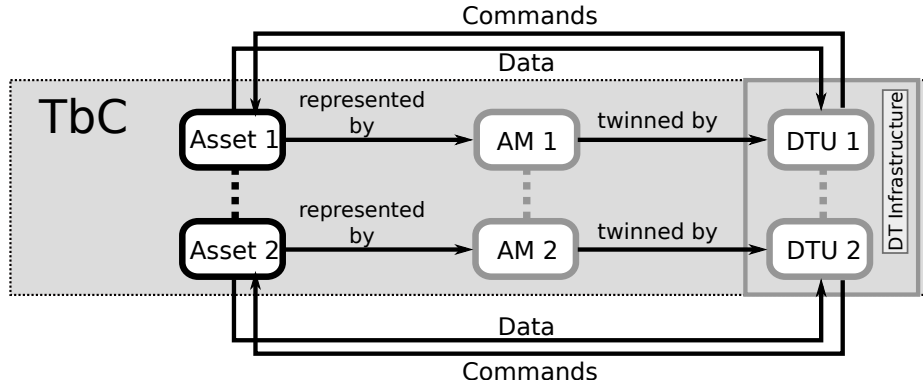
Fig. 2. Structure of a digital twin for two assets (Asset 1 and Asset 2) as PT units.

The Role of Knowledge Graphs. Knowledge graphs (KGs) are the established technique to formalize domain knowledge and add semantics (w.r.t. a domain) to data. In particular, they provide a uniform way to represent, query and reason about data. In the TbC approach, their role is twofold:

Uniform Structure Representation. The asset model can be a knowledge graph,[3] and the information required for twinning can be accessed using standard KG technologies. The DT infrastructure can be translated into a knowledge graph, using semantical lifting [4], that maps program states to KGs, meaning that the information about the structure of the DT has the same format and representation as the structure of the PT.

Uniform Data Access. Starting from the uniform representation of structure, both the user and the DT infrastructure itself can query and reason about the *combined* structure of DT and PT. The user can access the digital twin, e.g., its simulation results, in terms of the asset model. The DT infrastructure can detect structural drift between DT and PT, i.e., if the structural correspondence does not hold when querying over both structures.

3   Twinning-by-Construction and Temporal Properties

In the previous section we discussed how a given asset model, describing the state of a physical system, can be used to construct a digital system that is TbC. Now, we turn our attention to self-adaptation. Self-adaptation is triggered when we observe a deviation of the DT unit with respect to the PT unit and when there are changes in the asset model. When constructing self-adaptive systems in the context of digital twins, it is important to consider a library of reusable DT unit, support for a self-adaptation process that takes into consideration requirements for the reconstruction and support for reasoning, analysis, and tuning during the

---

[3] This approach is taken by several on-going projects, e.g., the READi [12] project.

reconstruction process [13]. This can be realised following the MAPE-K feedback loop schema [1,2,3] with four activities: Monitor, Analyse, Plan, and Execute, over a shared Knowledge.

In addition, we consider properties for both behavioural and structural relations to the assets. The reason for considering these additional properties is that while the physical systems change, the DT infrastructure must preserve the twinning property by reconfiguring the structure between the corresponding DT units within a time bound. This *structural* reconfiguration, which we call *re-twinning*, poses additional temporal constraints on the physical and digital systems, that twinning must be ensured at a certain speed. We now detail these different activities.

MAPE-K Schema for Data Streams. Self-adaptation is needed when we observe model drift: there is a deviation between a DT unit and its PT unit. In this context, the *monitor* step collects data streams from the physical and digital system. The *analysis* step can use temporal properties to express expectations on the data stream that are needed for the model in the digital twin to work, e.g, using runtime monitoring (see Fig. 5, top right monitor). In our building example, one may have a restriction that the moisture of the wall is never higher than 1%. If the sensors report a moisture value beyond this threshold, the system triggers an error. During the analysis step, one can also use a *hyperproperty* [14] and compare the data output from the physical system with the data output from the digital system to detect if the difference is above a threshold to trigger an error.

The *planning* step can use *model search techniques* to find the best parameters for the DT unit to adjust the observed output deviation from the PT unit, and the *execute* step will reconfigure the relevant DT units by resetting their parameters. This is a major application scenario for digital shadows,[4] which detect anomalies not only by analysing the data stream, but by the relation of data stream and the digital twin unit.

MAPE-K Schema for Structural Self-Adaptation. The connection of digital twin and asset model must also be maintained. As shown in Fig. 3, the self-adaptation process must *detect* structural changes in the asset, and trigger reconfiguration of the digital system such that the temporal TbC properties that were established initially holds again, as initially discussed in [7].

Example 2. Continuing with Example 1, we consider the situation that a third wall `as:w3` is added to the building under construction and, consequently, to the asset model. In the knowledge base, the following triples are added:

`as:w3 a as:Wall. as:w3 a as:InProd. as:w2 as:leftOf as:w3.`

---

[4] A digital shadow is a digital twin with unidirectional data flow: the DT does not send commands back to the PT [15].
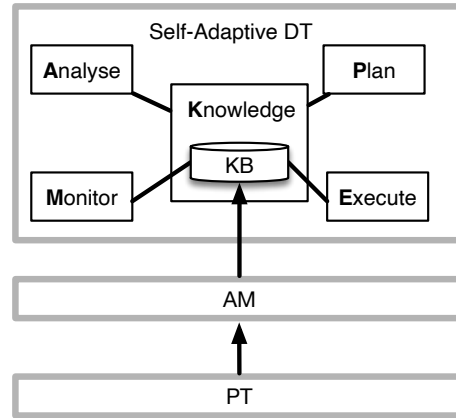
Fig. 3. Structural self-adaptation of a DT following a MAPE-K feedback loop.

Twinning uses the asset model, which is formalized as a knowledge base, to ensure that all assets that are to be twinned are indeed represented by some DT units and that the DT units are correctly connected (i.e., according to the PT). Additionally, TbC ensures that no DT units without a PT equivalent exist in the DT. It is application-specific what assets are modelled and how the DT infrastructure repairs its state – thus, it is not possible to generate a DT infrastructure fully automatically from an asset model.

However, in the *monitor* step we can formulate the condition that the DT is twinning the PT correctly by posing the query that returns the mismatches: (1) all assets not represented in the DT and (2) all objects not corresponding to an existing asset in the PT. An example of such a query is shown below. It assumes that the DT infrastructure can be seen as a knowledge base, where all its DT units are represented as instances of a class `dti:DTUnit`, and that each DT unit is connected to the asset it twins via the `dti:twins` property. The following SPARQL [16] query retrieves all assets that occur in the physical system but are not twinned.

```
SELECT ?x { ?x a as:InProd.
            FILTER NOT EXISTS (?y a dti:DTUnit. ?y dti:twins ?x.)
}
```

The *analysis* step is in charge of understanding that the structure is mirrored correctly, which can be expressed in our example simply by using `as:leftOf`. We can similarly define a query that returns all DT units that are not respecting the structure of the assets (shown in Fig. 4), where we assume that the structure of the DT infrastructure that connects DT units is using an analogous property `dti:leftOf`. We say that a system is *simply twinned* if both queries return an empty set. A *simply twinned* system has one DT unit for each PT unit, and all

```
SELECT ?dtu { ?dtu a dti:DTUnit. ?dtu dti:twins ?asset.
    OPTIONAL(
        ?asset as:leftOf ?right.
        FILTER NOT EXISTS (
           ?dtuRight a dti:DTUnit.
           ?dtu dti:leftOf ?dtuRight.
           ?dtuRight dti:twins ?right.
           )
    )
}
```

Fig. 4. A query to ensure that all twinned units are connected correctly. It returns the set of DT units that are *not* mirroring the structure correctly.

the DT units are correctly connected. Let `SIMPLE` be the conjunction of the two given queries.[5]

The *planning* step identifies the DT units that must be created; in this case, one DT unit for each member of the returned set in the queries, if non-empty.

The *execution* step creates the identified DT units and link them correctly to the existing DT units according to the asset model. The execution step triggers re-twinning, which is done according to the planning step.

Temporal Properties of the DT Infrastructure. It takes time to re-twin and it may be crucial to ensure that re-twinning happens sufficiently fast. For this reason, we also consider temporal properties of the DT infrastructure, which are here represented using Metric Temporal Logic (MTL) [17,18], an extension of LTL with intervals that is suited for online monitoring [19]. The syntax of a MTL formula φ over state predicates p and intervals I[6] is given by

$$\varphi ::= p \mid \textbf{false} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \, \mathbf{U}_I \, \varphi$$

and the abbreviations $\Box_I \, \varphi = \textbf{false} \, \mathbf{U}_I \, \varphi$, $\Diamond_I \, \varphi = \neg \Box_I \, \neg\varphi$, $\Box \, \varphi = \Box_{[0,\infty)} \, \varphi$. The intuition for $\Box_I \, \varphi$ is that φ holds at all points in time in the interval I and the intuition for $\Diamond_I \, \varphi$ is that φ holds at some point in the interval I.

As shown in Fig. 5, a property expressing that if a system is out of sync, then it will be re-twinned within n time units, is expressed as follows:

$$n{-}\mathsf{Resync} \equiv \Box\big(\texttt{SIMPLE} \neq \emptyset \rightarrow \Diamond_{[0,n]}\texttt{SIMPLE} \doteq \emptyset\big)$$

Observe that `SIMPLE` only works because there are several sources of data for the knowledge base: the domain knowledge, the asset model and the current state of the DT infrastructure.

---

[5] The example is given in SPARQL, but could also be formulated in, e.g., OWL as two classes that must be empty.

[6] We use discrete time with intervals of the form $[n, m]$ or $[n, \infty), n, m \in \mathbb{N}$.
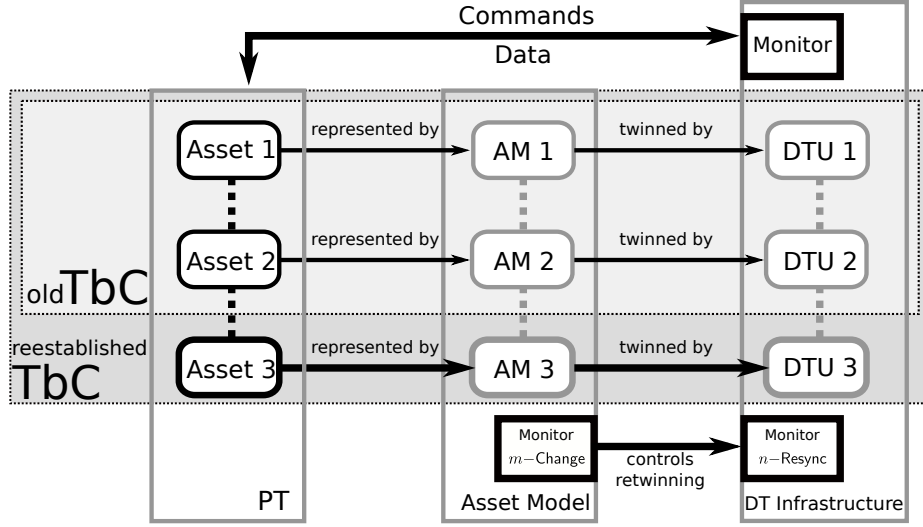
Fig. 5. Refined structure of the digital twin architecture after a change: an new DT unit is added for the new asset and the twinning property must be re-established.

Regulating the Rate of Re-Twinning. Twinning takes time. So one must make sure that the asset model is not changing too fast. For example, if every change in the physical system triggers re-twinning, then several connected changes (e.g., changing several assets in one maintenance) should be submitted at once to the asset model. Formally, one demands that a CHANGE is not happening more often than m time units. The following formula expresses this constraint.

$$m-\mathsf{Change} \equiv \Box\big(\mathtt{CHANGE} \to \neg\Diamond_{(0,m]}\mathtt{CHANGE}\big)$$

In our reference architecture, changes in the asset model and the re-twinning process can happen independently: $m-\mathsf{Change}$ monitors the asset model interface and reports if the asset model is updated too often, while $n-\mathsf{Resync}$ monitors the DT infrastructure itself, see Fig. 5.

The rate of change of the asset models, which captures the evolution of the PT, should not be faster than the twinning speed, i.e., the speed of the DT infrastructure that updates according to the changes in the asset model. Thus, the property to be monitored is

$$\mathtt{TEMP}_{m,n} \equiv m > n \wedge m-\mathsf{Change} \wedge n-\mathsf{Resync}$$

4   The Digital Thread as a Temporal Property

So far, we have considered properties that relate the current state of a digital system and the current state of the physical system, as well as temporal properties that ensure that twinning happens at the correct frequency. This, however, does

not consider the so-called "digital thread", which must take into account not only the current state of the system, but also its trace of past actions. This requires a specific notion of twinning for traces and has consequences for the structure of the DT infrastructure: not all DT units are relevant for the twinning property of the current state, and the DT units for replaced or removed assets must be handled differently. First, we describe an example of more involved changes in the physical system.

Example 3. Continuing Example 1, the newly built third wall (`as:w3` in the asset model) is used to *replace* the second wall (`as:w2` in the asset model), which is removed in the process. Afterwards `as:w3` is removed as well and `as:w1` remains as the sole wall. The total sequence of actions, the trace, in the physical system is as follows:

1. Build walls `as:w1` and `as:w2`
2. Build wall `as:w3`
3. Replace `as:w2` with `as:w3`
4. Decommission `as:w3`

The twinning actions in Steps 1 and 2 have been explained in Sec. 2 and 3. Now we continue from Step 3. The following asset model reflects this change by marking that the digital twin unit `as:w2` as decommissioned, `as:w3` as replacing `as:w2`. It updates the information that that `as:w1` is now directly left to `as:w3`.

```
as:w1 a as:Wall. as:w2 a as:Wall. as:w3 a as:Wall.
as:w1 a as:InProd. as:w2 a as:Decom. as:w3 a as:InProd.
as:w1 as:leftOf as:w3. as:w3 as:replaces as:w2.
```

The change on the asset model already realizes the digital thread. We can query about what the PT has performed from the knowledge base. For example, to retrieve the walls that `as:w2` was next to, we can run the following query

```
SELECT ?x {?x as:leftOf [as:replaces* as:w2]}
```

which gives us `as:w1`. The digital twin however, must keep the information about the DT unit for `as:w2` available as well. As we require that the DT infrastructure represents every physical asset that was part of the asset model at some point in a DT, it is guaranteed that such a DT unit existed, but it *must* still exist now and be ignored for the twinning property of the current state − all assets which are instances of the decommissioned class `as:Decom` must be ignored for it.[7] Thus, we refine the DT infrastructure and introduce the digital twin *graveyard*, see Fig. 6, a data structure where DT units of decommissioned, or otherwise removed, assets are saved, but ignored for the twinning of the *current* state.

The refined structure is shown in Fig. 6: The DT infrastructure contains a DT graveyard structure, where, compared to Fig. 5 one DT unit has been moved and is, consequently *ignored* for the twinning property. Note that the corresponding asset (Asset 1) has been removed from the physical system, but its asset model
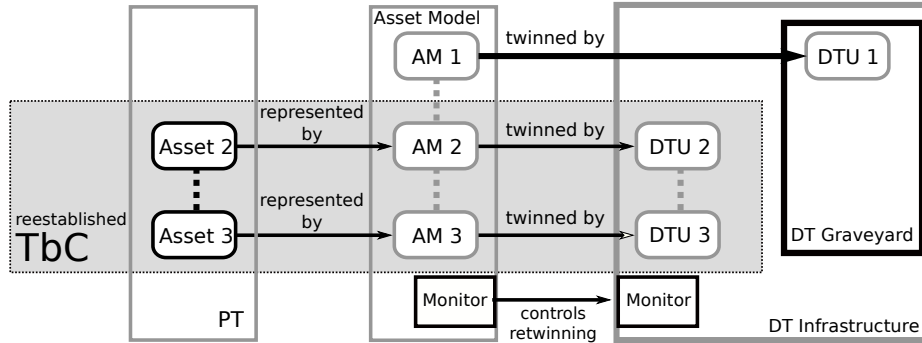
Fig. 6. Structure of the digital twin. The graveyard retains the DT units, which are needed for the digital thread, but must be ignored for the twinned property.

(AM 1) remains. The moved DT unit is also disconnected from the other DT units, while the asset model contains information about the old structure.

The digital thread is, thus, realised on the asset model (and possible connected to the original designs, requirements, etc.) of the walls, and in the DT infrastructure (using semantic reflection of the graveyard). However, there are further events in the trace of the physical system that are not changing the structure. For example, if a wall is repainted, it is visible in the asset model, but does not require a change in the DT. Similarly, changes in the DT infrastructure that are not structural, such as behavioural reconfigurations, must be made available to the user.

One can easily refine the asset model by recording more information about the events, for example their time and data and additional, explicit information when, how and why `as:w3` was replaced. We refrain from giving a realistic example using a full ontology based asset model like IMF [12], but illustrate the additional information by performing the last step on our building example.

Example 4. Next, we remove wall `as:w3` completely and we add information about the removal. The asset model becomes the following (in terms of Fig. 6, this is the complete asset model including `AM1`, `AM2`, and `AM3`, without general declarations of the classes).

```
as:w1 a as:Wall. as:w2 a as:Wall. as:w3 a as:Wall.
as:w1 a as:InProd. as:w2 a as:Decom. as:w3 a as:Decom.
as:w1 as:leftOf as:w3. as:w3 as:replaces as:w2.
as:w3 as:observed as:ev1. as:w3 as:removedAt as:ev2.
as:ev1 a as:Observation. as:ev1 a as:WaterDamage".
as:ev1 as:at "2022-05-21.12:10:00"^^xsd:date.
as:ev2 a as:Removal. as:ev2 as:byCompany "Parken".
as:ev2 as:at "2022-05-22.12:00:00"^^xsd:date.
```

---

[7] This is already modelled in the SIMPLE query above.

This marks `as:w3` as decommissioned, but also records two events: `as:ev1` records that there was some water damage at a specific date and `as:ev2` records that a removal was performed at a specific date by the company `Parken`. The fourth line of the asset model connects the `as:w3` with events `as:ev1` and `as:ev2`. The first event `as:ev1`, is not relevant for the structure of the DT infrastructure, yet may trigger a behavioural reconfiguration – our data stream monitor detects the violation of the moisture constraint, but the DT unit is not moved to the graveyard. Yet, the information of the reconfiguration is available in the knowledge graph of the DT infrastructure. For example, consider the following.

```
dti:dtu1 dti:twins as:w3.
dti:dtu1 dti:reconfiguration dti:ev3.
dti:ev3 dti:at "2022-05-21.12:11:00"^^xsd:dateTime.
dti:ev3 dti:newParam "1"^^xsd:int.
dti:dtu1 dti:removal dti:ev4.
dti:ev4 dti:at "2022-05-22.12:01:00"^^xsd:dateTime.
```

Now, the user can run queries to investigate possible causes for a reconfiguration, or the consequences of some event. For example, the following query returns all reconfigurations that happened on the date of a water damage:

```
SELECT ?reconf {
    ?asset as:observed ?ev. ?ev a as:WaterDamage.
    ?dtu dti:twins ?asset. ?dtu dti:reconfiguration ?reconf.
    ?ev as:at ?datetime. ?reconf dti:at ?datetime.}
```

A possible perspective is that we check whether an *event* on asset side has been twinned. However, a reconfiguration can mirror different events (or changes without events, due to model drift), so one cannot in general require a one-to-one correspondence between events in the DT infrastructure and events in the physical system.[8]

The above example shows the intricate interactions of asset models, traces, the re-establishment of twinning properties and self-adaptation at runtime. We now return to the expression of temporal properties and monitoring of twinning process itself.

Monitoring Traces. As discussed, consistency of the updating speed is guaranteed: Formula `TEMP` expresses that every change in the asset model is mirrored in time by the DT infrastructure and the digital twin is simply twinned again. For example, it cannot be the case that the DT updates too slowly and could not perform the addition of `as:w3` in Step 2 in time so that actions in Steps 3 and 4 are not able to be realised.

---

[8] Note that while we require a one-to-one correspondence between PT and DT units here, this is only a simplification for *simple* twinning, one can easily extend the system to handle one-to-many twinning relations.

A part of this is *correctness* w.r.t. simple twinning: Every physical unit (that is correctly handled in the asset model) has a DT unit twinning it. While the final state of the digital twin has the DT unit for `as:w1`, we know that the DT has also created DT units for the other assets. Next, we discuss how we formalize *temporal simple twinning*, which expresses simple twinning also for decommissioned physical units. And give the following query `TempSimple`:

```
SELECT ?x {
  ?x a dti:DTUnit.
  FILTER NOT EXISTS(
    ?x dti:twins ?asset. ?asset a as:Decom.
    ?asset as:removedAt [a as:Removal; as:at ?datetime1].
    ?x dti:removal [a dti:Remove; dti:at ?datetime2].
    FILTER (
      microsec(?datetime2) - microsec(?datetime1) < 5*60*1000
    )
  )
}
```

If the system is correctly twinned with respect to removal/replacement of PT and DT units, the query should return an empty set. So the query above returns all DT units, excepts those which have a removal event within 5 minutes of the removal of the corresponding asset they twin, i.e., we define 5 minutes as the time limit for re-twinning. It is guaranteed that this holds if (1) `TEMP` holds and (2) the DT infrastructure is implemented correctly. Thus, we monitor that the DT infrastructure correctly implements the following temporal property, *trace-based TbC*:

$$\text{TEMP}_{m,n} \wedge \Box\big(\texttt{TempSimple} \neq \emptyset \rightarrow \Diamond_{[0,n]}\texttt{TempSimple} \doteq \emptyset\big)$$

5   Related Work

The connection of digital twins and asset models so far is mostly used for data integration to handle the numerous heterogeneous data sources of the physical twin. For example, Yan et al. [20] use knowledge bases (KBs) to integrate data in manufacturing equipment, Banerjee et al. [21] use a similar approach to interact with data from IoT sensors in industrial production lines, and Oakes et al. [22] for drivetrains. Going one step further, Wascak et al. [23] aim to use asset models as part of this integration in their abstract digital twin architecture. More abstractly, Kharlamov et al. [24] have investigated the use of KBs for data integration in the context of the energy industry, and used this integrated data to enable machine learning on data streams [25]. Lietaert et al. [26] use KBs similarly to integrate data for machine learning approaches.

To the best of our knowledge, the use of KBs to handle structural drift of the PT and its asset model is hitherto unexplored. Various methods exist to detect parameter or model drift, both statistical, e.g. Woodcock et al. [27] and

formal, e.g. [28]. Multiple works have addressed formal modelling and verification of self-adaptive systems to provide assurances [29] using MAPE-K feedback loops. Various approaches, from very methodological such as ENTRUST [30] to more concrete, using formal techniques, such as ActivFORMS [31] that uses formal models at runtime in the form of timed automata. Recent works explore Petri nets to model self-adaptive systems, along with domain specific language [32,33,34]. Arcaina et al. have developed Abstract State Machines [35,3] to model interactive MAPE-K loops. Formal verification of MAPE-K loops have been also explored from the perspective of use cases. Feng et al. [36] adapts a case study for DT engineering to verify MAPE-K feedback loops and Päßler et al. [37] develops a formal model of Metacontrol [38] for the heating system of a smart home that considers MAPE-K loops and self-adaptation using a black box approach. Compared to the work presented in this paper, all the previous approaches focuses on the behavioural aspect of self-adaptation, and not on the structural aspect as discussed in this paper. To the best of our knowledge, we are not aware of other approaches that take into account structural self-adaptation and MAPE-K feedback loops.

6   Conclusion

This paper considers a digital twin infrastructure that combines MAPE-K feedback loops and semantic reflection. Whereas feedback loops can be used to realize runtime self-adaptation for the digital twin, semantic reflection enables the structure of the digital twin and of the asset to be uniformly represented in a knowledge base. We show how the resulting knowledge base can be queried by the digital twin application and reasoned over to detect misconfigurations that violate digital twin correctness properties. Consequently, the feedback loops and semantic reflection jointly ensure that the ensemble of digital artefacts is indeed always a correct twin, i.e., the combined digital artefacts are *twinned-by-construction*.

In the paper, we consider the monitoring of both structural and temporal correctness properties for digital twin reconfiguration, including that a structural twin relation is established by the repair function, that the time delay required by reconfiguration is within a given bound, and that the digital thread is reflected in the digital twin framework. To this aim, the knowledge base is extended to capture execution traces, such that the evolution of the physical and digital systems can be compared by querying the knowledge base.

In our current work, the correctness property of *twinning* is specified as an empty query result from a certain query on the knowledge base combining the asset model and the semantically reflected digital twin configuration.

In future work, we plan to investigate the feasability of our approach using a bigger case study that will be evaluated the SMOL framework, which supports both semantic reflection and digital artefacts embedded as FMUs. Furthermore, we plan to investigate the static verification of query-based specifications of programs with knowledge bases.

References

1. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H., Kienle, H.M., Litoiu, M., Müller, H.A., Pezzè, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. In: Software Engineering for Self-Adaptive Systems. Volume 5525 of Lecture Notes in Computer Science., Springer (2009) 48–70
2. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50
3. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing MAPE-K feedback loops for self-adaptation. In Inverardi, P., Schmerl, B.R., eds.: 10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2015), IEEE Computer Society (2015) 13–23
4. Kamburjan, E., Klungre, V.N., Schlatte, R., Johnsen, E.B., Giese, M.: Programming and debugging with semantically lifted states. In: Proc. 18th International Conference on the Semantic Web (ESWC 2021). Volume 12731 of Lecture Notes in Computer Science., Springer (2021)
5. Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., Viel, A.: Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: Modelica Conference, The Modelica Association (2012) 173–184
6. Kamburjan, E., Johnsen, E.B.: Knowledge structures for simulation units. In: ANNSIM. (2022) to appear.
7. Kamburjan, E., Klungre, V.N., Schlatte, R., Tapia Tarifa, S.L., Cameron, D., Johnsen, E.B.: Digital twin reconfiguration using asset models (2022) under review.
8. Gould, L.S.: What are digital twins and digital threads? Automotive Design & Production **23** (2018)
9. Margaria, T., Schieweck, A.: The digital thread in industry 4.0. In: IFM. Volume 11918 of LNCS., Springer (2019) 3–24
10. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman and Hall/CRC Press (2010)
11. W3C, RDF Working Group: Resource description framework https://www.w3.org/RDF.
12. Fjøsna, E., Waaler, A.: READI Information modelling framework (IMF). Asset Information Modelling Framework. Technical report, READI Joint Industry Project (March 2021) https://readi-jip.org/wp-content/uploads/2021/03/Information-modelling-framework-V1.pdf.
13. de Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B.R., Tamura, G., Villegas, N.M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R.J., Dustdar, S., Engels, G., Geihs, K., Göschka, K.M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovski, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., Pezzè, M., Prehofer, C., Schäfer, W., Schlichting, R.D., Smith, D.B., Sousa, J.P., Tahvildari, L., Wong, K., Wuttke, J.: Software engineering for self-adaptive systems: A second research roadmap. In de Lemos, R., Giese, H., Müller, H.A., Shaw, M., eds.: Software Engineering for Self-Adaptive Systems II -

Revised Selected and Invited Papers. Volume 7475 of Lecture Notes in Computer Science., Springer (2010) 1–32

14. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**(6) (2010) 1157–1210

15. Kritzinger, W., Karner, M., Traar, G., Henjes, J., Sihn, W.: Digital twin in manufacturing: A categorical literature review and classification. IFAC-PapersOnLine **51**(11) (2018) 1016–1022

16. W3C, SPARQL Working Group: Sparql 1.1 query language https://www.w3.org/TR/sparql11-query/.

17. Brandt, S., Kalayci, E.G., Kontchakov, R., Ryzhikov, V., Xiao, G., Zakharyaschev, M.: Ontology-based data access with a horn fragment of metric temporal logic. In: AAAI, AAAI Press (2017) 1070–1076

18. Koymans, R.: Specifying real-time properties with metric temporal logic. Real Time Syst. **2**(4) (1990) 255–299

19. Ho, H., Ouaknine, J., Worrell, J.: Online monitoring of metric temporal logic. In: RV. Volume 8734 of Lecture Notes in Computer Science., Springer (2014) 178–192

20. Yan, H., Yang, J., Wan, J.: Knowime: A system to construct a knowledge graph for intelligent manufacturing equipment. IEEE Access **8** (2020) 41805–41813

21. Banerjee, A., Dalal, R., Mittal, S., Joshi, K.P.: Generating digital twin models using knowledge graphs for industrial production lines. In: Proc. Web Science Conference (WebSci 2017), ACM (2017) 425–430

22. Oakes, B.J., Meyers, B., Janssens, D., Vangheluwe, H.: Structuring and accessing knowledge for historical and streaming digital twins. In Tiddi, I., Maleshkova, M., Pellegrini, T., de Boer, V., eds.: Joint Proceedings of the Semantics co-located events: Poster&Demo track and Workshop on Ontology-Driven Conceptual Modelling of Digital Twins co-located with Semantics 2021. Volume 2941 of CEUR Workshop Proceedings., CEUR-WS.org (2021)

23. Waszak, M., Lam, A.N., Hoffmann, V., Elvesæter, B., Mogos, M.F., Roman, D.: Let the asset decide: Digital twins with knowledge graphs. In: 19th IEEE International Conference on Software Architecture (ICSA 2022)

24. Kharlamov, E., Martín-Recuerda, F., Perry, B., Cameron, D., Fjellheim, R., Waaler, A.: Towards semantically enhanced digital twins. In: IEEE BigData, IEEE (2018) 4189–4193

25. Zhou, B., Svetashova, Y., Gusmao, A., Soylu, A., Cheng, G., Mikut, R., Waaler, A., Kharlamov, E.: Semml: Facilitating development of ML models for condition monitoring with semantics. J. Web Semant. **71** (2021) 100664

26. Lietaert, P., Meyers, B., Noten, J.V., Sips, J., Gadeyne, K.: Knowledge graphs in digital twins for AI in production. In Dolgui, A., Bernard, A., Lemoine, D., von Cieminski, G., Romero, D., eds.: Proc. IFIP WG 5.7 International Conference on Artificial Intelligence for Sustainable and Resilient Production Systems - (APMS 2021). Volume 630 of IFIP Advances in Information and Communication Technology., Springer (2021) 249–257

27. Woodcock, J., Gomes, C., Macedo, H.D., Larsen, P.G.: Uncertainty quantification and runtime monitoring using environment-aware digital twins. In Margaria, T., Steffen, B., eds.: Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part IV. Volume 12479 of Lecture Notes in Computer Science., Springer (2020) 72–87

28. Deshmukh, J.V., Donzé, A., Ghosh, S., Jin, X., Juniwal, G., Seshia, S.A.: Robust online monitoring of signal temporal logic. Formal Methods Syst. Des. **51**(1) (2017) 5–30

29. Weyns, D., Bencomo, N., Calinescu, R., Camara, J., Ghezzi, C., Grassi, V., Grunske, L., Inverardi, P., Jezequel, J.M., Malek, S., Mirandola, R., Mori, M., Tamburrelli, G.: Perpetual assurances for self-adaptive systems. In de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., eds.: Software Engineering for Self-Adaptive Systems, Cham, Springer (2017) 31–63

30. Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M.U., Habli, I., Kelly, T.: Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Transactions on Software Engineering **44**(11) (2017) 1039–1069

31. Iftikhar, M.U., Weyns, D.: ActivFORMS: Active formal models for self-adaptation. In: Proc. 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014), ACM (2014) 125–134

32. Fakhir, M.I., Kazmi, S.A.R.: Formal Specification and Verification of Self-Adaptive Concurrent Systems. IEEE Access **6** (2018) 34790–34803

33. Mian, N.A., Ahmad, F.: Modeling and Analysis of MAPE-K loop in Self Adaptive Systems using Petri Nets. International Journal of Computer Science and Network Security (IJCSNS) **17** (2017) 6

34. Camilli, M., Capra, L.: Formal specification and verification of decentralized self-adaptive systems using symmetric nets. Discrete Event Dynamic Systems **31**(4) (December 2021) 609–657

35. Arcaini, P., Riccobene, E., Scandurra, P.: Formal Design and Verification of Self-Adaptive Systems with Decentralized Control. In: ACM Transactions on Autonomous and Adaptive Systems. (2016)

36. Feng, H., Gomes, C., Gil, S., Mikkelsen, P.H., Tola, D., Larsen, P.G., Sandberg, M.: Integration of the MAPE-K loop into digital twins ANNSIM'22, To appear.

37. Päßler, J., Aguado, E., Silva, G.R., Corbato, C.H., Johnsen, E.B., Tapia Tarifa, S.L.: A formal model of Metacontrol in Maude (2022) under review.

38. Corbato, C.H.: Model-based self-awareness patterns for autonomy. PhD thesis, Universidad Politécnica de Madrid (2013)