# Digital Twin Reconfiguration Using Asset Models[*]

Eduard Kamburjan, Vidar Norstein Klungre, Rudolf Schlatte, S. Lizeth Tapia Tarifa, David Cameron, and Einar Broch Johnsen

Department of Informatics, University of Oslo, Oslo, Norway
{eduard,vidarkl,sltarifa,rudi,einarj,davidbc}@ifi.uio.no

**Abstract.** Digital twins need to adapt to changes in the physical system they reflect. In this paper, we propose a solution to dynamically reconfigure simulators in a digital twin that exploits formalized asset models for this purpose. The proposed solution uses (1) semantic reflection in the programs orchestrating the simulators of the digital twin, and (2) semantic web technologies to formalize domain constraints and integrate asset models into the digital twin, as well as to validate semantically reflected digital twin configurations against these domain constraints on the fly. We provide an open-source proof-of-concept implementation of the proposed solution.

## 1 Introduction

Digital twins are model-centric applications in which some asset — typically a physical system — is mirrored in near real-time, or *twinned*, by a digital artefact in order to understand, predict or control the behaviour of the asset (e.g., [15,36]). We envision a digital artefact, the so-called *digital twin* (DT), that contain components that compare the behaviour of the targeted asset, the so-called *physical twin* (PT), with expected behaviour based on a model, optimize the behaviour of the asset and prototype new designs. A typical example of such a component in the digital twin is a simulation model to explore the expected behaviour of the physical system.

The digital and physical twins are *coupled*: Data concerning the physical twin, such as live sensor data obtained by monitoring the physical system, are transmitted to the digital twin. Decisions made by the digital twin by analysing this data in the context of its model of the physical system, are communicated back. The connection between the digital twin and the physical system, and the integration of new observations of the physical system (such as live sensor data) into the digital twin's model of the physical system, are realized by the application itself; in this work, we refer to this architectural layer of the digital twin as the *Digital Twin Infrastructure (DTI)*; The DTI is not only responsible

[*] This work was supported by the Research Council of Norway through the projects SIRIUS (237898) and PeTWIN (294600).

for communication between the components of the DT but also of its evolution through the dynamic reconfiguration of simulation models.

As the physical system evolves over time, we may experience that the digital twin and the physical twin drift apart. This leads to a precision loss in the digital twin's ability to reflect the behaviour of the physical system. For example, the physical system may change through maintenance operations or unexpected events (such as failures). The simulation model may also drift due to uncertainty in parameters or noise in the sensor data it receives from the physical system.

This paper considers how digital twins can be *dynamically reconfigured* in response to changes in the physical twin. There are two categories of reconfiguration. The first category is *behavioural reconfiguration*, where the behaviour of the digital twin must be adapted but the structure of the physical system remains intact. For example, if simulated behaviour drifts away from the sensor data from the physical system, the corresponding simulator must be recalibrated with different parameters to match the real behaviour [8]. The second category is *structural reconfiguration*, where the structure of the physical system changes and the digital twin must perform some adaptation that goes beyond adjusting a single component. For example, reconstruction work or reorganisation in a factory may affect its entire production pipeline. The main focus of this paper is on the structural reconfiguration of digital twins.

To accommodate such reconfigurations, we here consider the use of *semantic web* technologies to connect the configuration of the digital twins to formalized *asset models*. Semantic web technologies are logic-based techniques to formalize knowledge and data, as well as to query and reason over the formalized knowledge represented as a *knowledge graph*. Semantic web technologies have been recognized as one of the potential pillars in symbolic AI for digital twins. Asset models are descriptions of the composition and properties of some physical asset, which is essential to represent the structure of the physical system not only for digital twins, but also for other engineering and maintenance applications [32,41].

In short, the main contributions of this paper are:

- a solution for structural reconfiguration of digital twins using formalized asset models, and
- a proof-of-concept realization of a digital twin infrastructure which orchestrates and configures simulation models, integrated with asset models.

We use the *Semantic Micro Object Language (SMOL)* [21] for our implementation. SMOL is a small, experimental and formally defined programming language with explicit primitives both to integrate simulation units, namely for the Functional Mock-Up Interface [4], and to integrate semantic web technologies that operate directly on the program state of the SMOL program itself. SMOL is open source and available from `http://smolang.org`.

*Related Work.* The connection of digital twins and knowledge bases so far is mostly limited to data integration to handle the numerous heterogeneous data sources in a digital twin. For example, Yan et al. [43] use knowledge bases to integrate data in manufacturing equipment and enable the user to query this

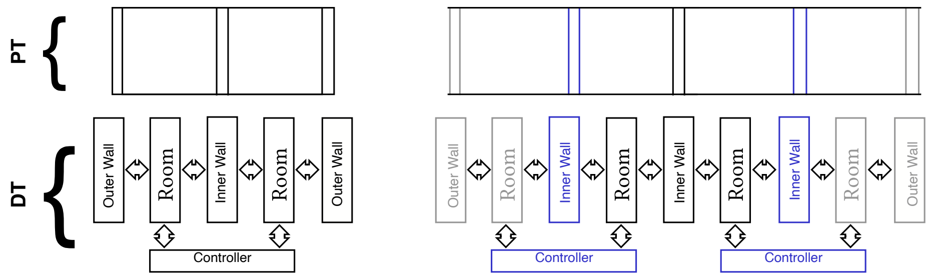

**Fig. 1.** A house (left top), its digital twin (left bottom), an extension of the house (right top) and the corresponding structural reconfiguration of the digital twin (right bottom). In the reconfigured digital twin, gray components are new and blue components need to be adapted.

information more easily. Banerjee et al. [2] use a similar approach to interact with data from IoT sensors in industrial production lines, and Oakes et al. [28] for drivetrains. Going one step further, Wascak et al. [40] aim to use asset models as part of this integration in their abstract digital twin architecture. More abstractly, Kharlamov et al. [22] 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 [45]. Lietaert et al. [25] use KBs similarly to integrate data for machine learning approaches. To the best of our knowledge, the use of KBs to influence the structure of the digital twin after its initial construction is hitherto unexplored. We discuss related work for asset models in Sec. 3.

*Structure.* Section 2 explains the problem of digital twin reconfiguration in terms of a motivating example and Sec. 3 introduces preliminaries. We use the motivating example to discuss the interplay of asset models, semantic web technologies, simulation units and programming in Sec. 4 and structural reconfiguration in Sec. 5. Section 6 concludes the paper.

## 2 Motivating Example

Let us consider the digital twin of a small house, which should retain some targeted temperature (inspired by an example developed by OSP [34]). The physical system consists of two rooms, each with an outer wall and a heater, separated by an inner wall. The physical system is depicted in Fig. 1 (top left). The corresponding DT has five simulators modelling the rooms with their heaters, the inner wall and the outer walls, respectively. In addition, the DT includes a controller that decides how to regulate the heaters of the two adjacent rooms, based on the input data. The DT is depicted in Fig. 1 (bottom left). In our example, the controller's restriction to two adjacent rooms is inherent to the available controller software.

```
@prefix asset:<https://smolang.org/Asset#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd:<http://www.w3.org/2001/XMLSchema#>.

asset:wall1 rdf:type    asset:Wall.
asset:wall1 asset:id    "2"^^xsd:string.
asset:wall1 asset:left  asset:room1.
asset:wall1 asset:right asset:room2.
```

**Fig. 2.** Example RDF graph with four statements.

A *behavioural reconfiguration* of the digital twin could be triggered by the following scenario: One of the heaters breaks down and is replaced with a different model which heats the room faster than the previous heater. The digital twin needs to reconfigure itself to adjust the parameters of its corresponding simulation model to reflect that heating in one of the rooms now works faster.

A *structural reconfiguration* of the digital twin could be triggered by the following scenario: The house is extended with two new rooms, one to the left and the other to the right of the existing rooms. The resulting physical system is depicted in Fig. 1 (top right), where the blue walls depict the former outer walls of the house, which have now become inner walls. This change in the physical system would require a complete reconfiguration of the digital twin. A solution, depicted in Fig. 1 (bottom right), would be to add four new simulators which capture the new rooms and new outer walls of the physical system (gray color in the figure), to change two of the existing simulators to reflect the change from outer walls to inner walls (blue color in the figure), and to replace the old controller by two new controllers (blue color in the figure). For example, the old controller could be removed and two new controllers added to reflect the constraint that only two adjacent rooms can be controlled by one controller.

## 3 Preliminaries

This section covers technical background for the proposed digital twin infrastructure: Knowledge bases and semantic web technologies, asset models and simulation units.

### 3.1 Knowledge Bases

Knowledge bases are a triple-based data representation of domain knowledge and other axioms. We here ignore their theoretical properties as description logic models, and briefly introduce four essential semantic web technologies: RDF, OWL, SPARQL and SHACL. All these technologies are W3C recommendations.

The *Resource Description Framework* (RDF) [38] is the framework and data model on which all other Semantic Technologies are built. RDF is used to describe entities (called resources) and their relation to other resources and data

```
1 asset:Room rdf:type owl:Class ;
2   rdfs:subClassOf [ rdf:type owl:Restriction ;
3                     owl:onProperty asset:next ;
4                     owl:cardinality 2 ].
5 asset:next rdf:type owl:ObjectProperty.
6 asset:next rdfs:domain asset:Room.
7 asset:next rdfs:range asset:Wall.
8
9 asset:room1 asset:next asset:wall5
```

**Fig. 3.** Example OWL ontology.

values (called literals). Each such relationship can be represented as a link between a subject and an object via a given predicate. In RDF, triples of the form (*subject*, *predicate*, *object*) are called statements, and multiple statements constitute a knowledge graph. Each RDF resource is represented by a Uniform Resource Identifier (URI), a unique identifier which makes it possible to refer to the same resource in different RDF graphs.

Figure 2 shows an example RDF graph using Turtle[1] syntax. The first three lines are all prefix declarations (aliases) while each of the lines 5–8 corresponds to a single triple. The statement in line 5 expresses that an entity `asset:wall1` exists and that it is a wall. The statement consists of a subject that is a resource with the URI `https://smolang.org/Asset#wall1`, which has been simplified to `asset:wall1` by using the prefix in line 1, a predicate that is a resource with URI `rdf:type` that expresses type membership, and an object `asset:Wall`, which is an OWL class.[2] The next line gives an identifier to the wall (`asset:id`), in this case the value `"2"` of type `xsd:string`. The last two lines express that the wall is between the two rooms `asset:room1` and `asset:room2`.

*OWL* [37] is an extension to RDF that makes it possible to develop complex models (called ontologies) of any application domain. OWL provides a vocabulary to declare which classes and properties exist, and the rules to which each such class and property must adhere. Figure 3 shows how OWL can be used to make a small ontology for the house example from Sec. 2. The statements in lines 1–4 declare an OWL class `asset:Room`, that has exactly two things stored using the `asset:next` Property. OWL has the open world assumption, at this point we know there are 2 things next to a room, but we may not have them explicit in our KB. The property `asset:next` itself is defined in lines 5–7 and relates `asset:Room` instances to `asset:Wall` instances. In line 9, we increase the KB and said that one of the things next to a `room1` is `wall5`.

[1] `https://www.w3.org/TR/turtle/`

[2] OWL classes and individuals are declared when they occur in a triple, not in a separate construct. We can derive that `asset:Wall` is a class, because it is a subject of a triple with predicate `rdf:type`. One can add a triple `asset:Wall a owl:Class` to make this explicit.

OWL gives precise and formal semantics to RDF, which allows automated reasoners to detect inconsistencies in RDF graphs and to infer implicit facts. For example, by considering the range and domain of the property `asset:next`, a reasoner can infer that the individual `asset:room1` in line 9 is an `asset:Room` and the individual `asset:wall1` is a `asset:Wall`. Subproperties can be defined in OWL; for example, we can declare property `asset:left` to be a subproperty of `asset:next`, in the sense that all `Room` resources that are `left` of a `Wall` resource are also `next` to that `Wall` resource.

*SPARQL* [30] is the prevailing language for querying and manipulating RDF graphs. It resembles languages such as SQL, but the way it specifies which data to return is specially suited for RDF graphs. In SPARQL, a basic `SELECT` query contains a `WHERE` clause with a graph pattern, which is an RDF graph where parts of the pattern are replaced with variables. The answers to this query are all subgraphs of the RDF knowledge graph that match this pattern, as given by the values from the RDF knowledge graph that were assigned to the variables in the matched pattern. This is demonstrated in the following query, which asks for all walls and their id.

```
SELECT ?a ?i WHERE{?a rdf:type asset:Wall. ?a asset:id ?i}
```

Here, `?a` and `?i` are both variables. When this query is posed over the RDF graph in Fig. 2 it returns only one result: `?a = asset:wall1, ?i = "2"`. SPARQL queries can access derived information by means of a logical consequence relation (technically, an *entailment regime*) [11,12,23].

In contrast to SPARQL, *SHACL* [39] ignores information that is not explicit in the knowledge base. SHACL is not used to check consistency of the knowledge base with respect to the ontology, but to ensure basic validity conditions on the concrete data.

## 3.2 Asset Modelling

An asset model is an organized description of the composition and properties of an asset (e.g., [16,32,41]). It is common practice in engineering to build asset models to support, e.g., maintenance operations on an asset. Asset models are useful in a digital twin context because they can provide the twin with static configuration data for the twin's simulation model [6]. In industrial applications, the asset model is often spread across several databases such as an asset management system, a engineering database and computer-aided design systems. Asset models may be directly formalized as knowledge bases (e.g., READI [9]), connected to them [26], or treated as such by means of ontology-based data access [29,33] which enable data integration across multiple databases. We can conceptually distinguish two kinds of asset models: top-down and bottom-up.[3]

*Top-down asset models* start with modelling the desired functionality of the cyber-physical system as a whole, and then decomposing the system into functional sub-systems. There is a tight coupling between functional sub-systems

[3] Standard semantic data for both top-down and bottom-up asset models is the subject of current research projects (e.g., DEXPI [42], CFIHOS [18] and READI [9]).

and simulation components in the digital twin setting [35,27]. This approach, which relates to model-driven engineering [3], is supported by modelling tools and languages such as SysML (e.g., [27]). A top-down model provides a scalable framework for tracking requirements along a system decomposition and linking requirements to individual components to higher-level system requirements [7,10]. However, the semantics of top-down models can be less well defined than for bottom-up models; for example, the Reference Designation System for ISO/IEC81346 [17,31] provides a taxonomy of functional systems for engineering, energy and construction, but is not so far supported by an ontology.

*Bottom-up asset models* can be given a well-defined semantics since they are organized around the actual physical components of the asset. Depending on the domain, the semantics and data models are provided by standards like ISO15926 [24] for the process industries, ISO10303 for manufacturing and aerospace [1] and BIM [44,5] for the built environment. The common feature of these models is their focus on physical artefacts. The functional behaviour that we want to simulate, is modelled by functional objects that correspond to the physical object. This tight linkage to the physical artefact means these models do not scale well for managing information about system behaviour and requirements, even though they are effective in organizing detailed information regarding individual components.

For the purposes of this paper, we work with a bottom-up asset model. We do this because, for this simple example, there is a tight correspondence between physical artefacts and systems. We do not commit to any specific standard and instead use an RDF based representation and our own ontology. This corresponds to the abstraction layer one would typically use after ontology-based data integration of the different databases that make up a complex asset model. Aspects typically associated with top-down asset models, such as the connection to the functionality, are here realized by the digital twin infrastructure.

### 3.3 Simulation Units

Simulation units are simulator prototypes that can be instantiated as simulation instances to perform some computation. Simulation units have inputs (to influence the computation) and outputs (to access results) and perform the computation step-wise, where the step size is determined by the driver that uses the simulation instance.

Formally, simulation units [13] are hextuples $(S, U, Y, \mathtt{set}, \mathtt{get}, \mathtt{doStep})$, where $S$ is the internal state space, $U$ the set of input variables, $Y$ the set of output variables, $\mathtt{set} : S \times U \times \mathcal{V} \rightarrow S$ the function to set the values of the input variables to some values of domain $\mathcal{V}$, $\mathtt{get} : S \times Y \rightarrow \mathcal{V}$ the function to get the results and $\mathtt{doStep} : \mathtt{S} \times \mathbb{R}^+ \rightarrow \mathtt{S}$ the function to perform the simulation for a given amount of time.

We work with a special form of simulation units, namely *functional mock-up units* (FMUs) [14], as defined by the functional mock-up interface (FMI) [4]. The FMI defines additional structures for simulation units, such as types or parameter variables, which cannot be reset, and additional information on the correct usage, e.g., the order of calls needed to initialize an FMU. Most importantly, it defines the *model description*, an XML formatted description of input and output variables, and further information about the FMU.

## 4 Semantically Lifted Co-Simulation

Semantically lifted programs can interpret their runtime state as a knowledge base, and access this knowledge base *describing their own current state* by means of language primitives at runtime [21]. This *semantic reflection* operates on a knowledge base which connects the representation of the runtime state with further ontologies, most importantly static domain knowledge and, in our case here, asset models.

Semantic lifting is supported by the Semantic Micro Object Language (`SMOL`), a Java-like object-oriented programming language. Semantic reflection in `SMOL` is realized through dedicated language primitives, such as an **`access`** expression, which loads the result of a SPARQL query, evaluated over a lifted enriched ontology under a logical consequence relation (see Sec. 3.1), into a list of objects in the runtime state.

Beyond semantic lifting, `SMOL` supports *Functional Mock-Up Objects*, a transparent layer that tightly integrates FMUs directly into the object model [19]. We here introduce `SMOL` by presenting a minimal digital twin infrastructure for the first scenario of Sec. 2 (see Fig. 1 left) and highlight its distinctive features as we proceed.

*Digital Twin.* Let us now consider a digital twin of the house from the motivating example in Sec. 2. The overall structure of the digital twin consists of objects of the classes `Wall` and `Room` that mirror the structure of the physical house, and objects of the additional classes `House`, that manages the overall DTI of a single house, `Outside`, to provide the context for the house, and class `Controller`, to make decisions about the heating behaviour.

Figure 4 shows the classes `Dynamic` and `Wall` (the class `Room` is analogous to `Wall`). In the figure, the class `Dynamic` defines two methods that are used for co-simulation (`propagate` and `advance`) to propagate values and uniformly advance time throughout the system, and a getter (`getHeat`) to access the output heat of a simulator. The class `Wall` defines a wrapper for the simulation unit of a wall, it has the following four fields:

- The field `fmo` contains an FMO, i.e., a wrapped FMU. Its type `Cont` wraps descriptions of the ports to the FMU: there are two input variables (`T_room1` and `T_room2`), preceded by the modifier **`in`**, and one output variable (`h_wall`), preceded by the modified **`out`**.
- The fields `left` and `right` point to the areas to the left and the right of the wall (an area is either a room or the outside).
- The field `id` is the identifier in the asset model for the physical wall that this object is mirroring (see Fig. 2).

```
 1 abstract class Dynamic()
 2   abstract Int propagate()
 3   abstract Int advance(Double db)
 4 end
 5
 6 class Wall extends Dynamic(
 7   Cont[in Double T_room1, in Double T_room2, out Double h_wall] fmo,
 8   Area left, Area right, Int id)
 9
10   override Int propagate()
11     this.fmo.T_room1   = this.left.getHeat();
12     this.fmo.T_room2   = this.right.getHeat();
13     return 0;
14   end
15   override Int advance(Double db)
16     this.fmo.tick(db); return 0;
17   end
18   Double getHeat() return this.fmo.h_wall; end
19 end
```

**Fig. 4.** SMOL class for the digital twin of a wall.

The methods realize the propagation of values into the FMU, time advance and the reading of the current temperature of the wall. In SMOL, the FMO is treated as a standard object with a method for time advance (l. 16) and the variables of its type are treated as fields (e.g., l. 11).

We next discuss how to instantiate a wall according to an asset model. Before introducing the asset model for our example in full detail in Sec. 5, we consider a restricted form here: There are OWL classes `asset : Wall` and `asset : Room` that model physical walls and rooms, with a property `asset : id` for their id and two properties `asset : left` and `asset : right` that connects a room to the wall left and right of it. Additionally, there is an OWL subclass `asset : Outer` of `asset : Wall` for the outermost walls.

The code in Fig. 5 shows how `Wall` instances for the outermost walls are created by exploiting the semantic reflection. This requires both access to the semantically lifted program state (to query over the ids of existing objects) and an external knowledge base with an asset model (to query over the ids of existing walls). First, a SPARQL query is executed on the knowledge base (l. 3), using the **access** statement, to select all the ids of outermost walls from the physical asset, from this list we retain the ids of outermost walls which are not stored in the `id` field of any existing `Wall` object (l. 4). Then, a new FMO is loaded for each id, using the **simulate** statement which takes the filepath to an FMU file (l. 7). The type of the FMO is then checked against the variable description given in the model description of the FMU file. Finally, the `Wall` object itself is created and stored in the list of all outermost walls.

```
List<Wall> walls = null;
List<Int> outer
  = access("SELECT ?id WHERE {?a asset:id ?id. ?a a asset:Outer.
              FILTER NOT EXISTS { ?o prog:wall_id ?id}}");
for Int i in outer do
   Cont[in Double T_room1, in Double T_room2, out Double h_wall] fmo
     = simulate("outerWall.fmu");
   walls = this.add(walls, new Wall(fmo, null, null, i));
end
```

**Fig. 5.** Prettified `SMOL` code loading walls from the asset model into the digital twin.

Note that the connection of the `outerWall.fmu` FMU and the `asset : Outer` OWL class is established by the digital twin infrastructure, i.e., the code creating the object instances, as we assume a bottom-up asset model. Furthermore, communication to the physical system happens through the `Room` objects, which encapsulate some interface to push and pull values to (resp. from) the physical system. The actual control of the actuators for the heaters in the `Room` objects is not detailed in this paper.

*Behavioural Reconfiguration.* Reconfiguration can be either structural or behavioural. A behavioural reconfiguration does not change the structure of the DT or DTI, but reacts to changes in the data stream from the PT, such as detected model/sensor drift. To do so, parameters of the existing twinned structure must be set again. Similarly, newly created DTs must be configured as part of their initialization. In the example above, where an outer wall is loaded, it does not suffice to simply create an FMU, if the FMU is used for simulation and not only as an interface to the PT.[4]

Consider an FMU `outerSim` and the case where the outer wall FMU has an additional parameter `in Double p`, which may be set to some value in the interval $[-1, 1]$, and a starting point `in Double init`. To estimate this parameter, one may collect some additional data and perform a model search, for example by recording the $n$ data points coming from the PT, and then testing which value for `p` generates the best fit for these data points. Fig. 6 shows a simple linear search for this case [19].

In case an FMU is replaced, the state of the old FMU, which may not be fully exposed, may contain additional information required for the simulation. To handle this, either the FMUs must expose enough information about their inner state to allow such operations, or the parameters of the new FMU, if there are any, must be determined.

Behavioural reconfiguration must be part of structural reconfiguration as the sensor streams, simulators and eventual feedback communication units are all

[4] If it is used as an interface, the identifier of connection to the PT must be given to the FMU (this is elided here).

```
Cont[...] reconfigure(Double last, List<Double> sysVal, Int n)
 Double step = -1;
 List<Double> sim = null;
 while step <= 1 do
  Cont[...] wall = simulate("outerSim.fmu", init=last, p=step);
  for( 0 <= i <= n ) do
   wall.tick(1.0);
   sim = Cons(shadow.h_wall, sim);
  end
  Double d = compare(sim, sysVal); //some error measurement
  if(d <= threshold) then return wall; end //new parameter
  step = step + 0.1;
 end
 return null; //no parameter found
end
```

**Fig. 6.** Model search for behavioural reconfiguration.

affected by changes in the asset, but as the mechanisms for it are orthogonal to asset models, we refrain from discussing it in more detail.

## 5 Structural Reconfiguration in SMOL

In this section, we consider the structural reconfiguration of the digital twins, focusing on its simulation component. For the digital twin infrastructure to structurally reconfigure its simulation component, we must (1) detect that the Digital Twin and Physical Twin have structurally drifted apart, as well as the exact kind of change that has occurred, (2) amend the relation between DT and PT, and (3) repair the Digital Twin Infrastructure. We continue with our house example to illustrate how semantic reflection is used to detect structural drift and monitor basic properties after repair. In this paper, we consider a domain-specific approach to the problem of structural reconfiguration in which the reconfiguration of the DT mimics the changes that occur in the PT.

Recall from Sec. 2 that a house must have an even number of rooms to be twinnable. For the sake of the example, we thus assume that the only structural changes that can occur to the asset is adding two rooms to the left of the existing rooms, adding two rooms to the right of the existing rooms, or adding one new room to the left and one to the right of the existing rooms in the house.

*Detecting Structural Drift.* Every $n$ simulation steps, the DTI runs a query to retrieve all IDs of rooms and walls that are in the asset model but not in the DT. If the number of such IDs is neither 0 (no change) or 2 (valid change), then the change is rejected – it is expressing an update that is not possible to twin because it violates our assumptions about the asset model and its changes.

```
class RoomAssert(String room, String wallLeft, String wallRight) end
....
List<RoomAssert> newRooms =
  construct("
  SELECT ?room ?wallLeft ?wallRight WHERE
  { ?x a asset:Room;
       asset:right [asset:Wall_id ?wallRight];
       asset:left [asset:Wall_id ?wallLeft]; asset:Room_id ?room.
    FILTER NOT EXISTS {?y a prog:Room; prog:Room_id ?room.} }");
if newRooms != null then // if newRooms == null then no update is needed
  Int nrRooms = newRooms.length();
  if nrRooms != 2 then /* report error */ end
  RoomAssert n1 = newRooms.content;
  RoomAssert n2 = newRooms.next.content;
  ...// (continued in Fig. 8)
end
```

**Fig. 7.** Detecting structural drift using semantic reflection.

The relevant query is given in Fig. 7. It constructs `RoomAssert` instances, each containing the room id of the room in the asset model, as well as the ids of the walls to the left and the right of the room. The query itself is analogous to the example described in Fig. 5. Afterwards, the number of retrieved rooms is used to detect whether a change has happened and, if so, whether the resulting house is still twinnable. The new house is twinnable if there are two new rooms that satisfy our criteria (see above).

Next, the position of the new rooms with respect to the existing structures needs to be detected. To this aim, we determine whether the two new rooms are adjacent to each other, their spatial relation to each other, and their relation to the left-most (resp. right-most) existing room. This is shown in Fig. 8: the first case is that the first retrieved room (`r1`) is right of the second room (`r2`) and left of the existing structure. The second case is that the first retrieved room (`r1`) is left of the second room (`r2`) and right of the existing structure. The last two cases are when the two new rooms are not adjacent and the remaining cases are omitted for readability.

*Structural Reconfiguration.* Having detected the kind of structural drift, the structure of the DT can be updated in two steps. First, we create the new simulation elements and insert them into the structure. Second, we update the DTI and repair possible virtual elements that are not reflecting elements in the asset (such as the controllers in our example). Figure 9 shows the resulting method which implements the addition of one new room to each side of the existing model. First, the rooms are created using `addOneLeft` and `addOneRight`, then the controller structure is rebuilt in `rebuildCtrl`, before we finally use SHACL to validate that the structural constraints hold for the new model configuration.

```
if n1.wallLeft == n2.wallRight &
  n1.wallRight == house.firstRoom.wallLeft.id then
  house.addTwoRight(n1.wallLeft, n1.room, n2.wallLeft, n2.room);
else
...
if n1.wallRight == n2.wallLeft &
  n1.wallLeft == house.firstRoom.wallRight.id then
  house.addTwoLeft(n1.wallRight, n1.room, n2.wallRight, n2.room);
else
...
if house.firstRoom.wallLeft.id == n1.wallRight then
  house.addLeftRight(n1.wallLeft, n1.room, n2.wallRight, n2.room);
else
  house.addLeftRight(n2.wallLeft, n2.room, n1.wallRight, n1.room);
end
...
```

**Fig. 8.** Determining the kind of structural drift.

```
Unit addLeftRight(String iw1, String ir1, String iw2, String ir2)
  this.addOneLeft(iw1, ir1); this.addOneRight(iw2, ir2);
  this.firstRoom.rebuildCtrl();
  Boolean valid = validate("examples/House/shape.ttl");
  if !valid then /* report error */ end
end
```

**Fig. 9.** Adding two rooms and reconstructing the controller structure.

Figure 10 details the addition of a single room. Object creation is straightforward, the interesting change is at its end where the old outer wall becomes an inner wall and the method reloads the FMU. An (omitted) method `calibrate` can adjust the behavioural configuration of the newly loaded FMU, if needed. If the FMU is only an interface that receives data from the PT, then this method may not perform any action. The method `addOneRight` has no counterpart in sole operations on the PT (i.e., addition of a single room is not supported), thus is does not validate the structure at its end.

Figure 11 details the code to completely rebuild the controller structure. It is called on the left-most room and creates a new controller, connects it to the currently considered room and its neighbour, deletes the old controller and continues with the next room with an old controller. A particular detail of semantic lifting is that it requires manual memory management: objects are retrievable by queries even if no pointer to them exists and can, thus, not be garbage collected.

*Validation.* We can use the knowledge base also to validate consistency constraints. This can be done either using the underlying logic, e.g., by checking

```
Unit addOneRight(String idw, String idr)
 //create new wall and room
 Cont[...] new_outer = simulate("examples/DummyFMUs/OuterWall.fmu");
 Wall new_wall  = new Wall(idw, new_outer, null, null);
 Cont[...] new_room_fmu = simulate("examples/DummyFMUs/Room.fmu");
 Room new_room =
   new Room(idr, new_room_fmu, null, null, null, False, null);
 new_wall.areaLeft = new_room; //link
 ...
 //repair old outer wall
 Cont[...] new_inner = simulate("examples/DummyFMUs/InnerWall.fmu");
 new_room.wallLeft.fmuSim = new_inner;
end
```

**Fig. 10.** Adding one room and adjusting the wall simulator.

```
Unit rebuildCtrl()
  Cont[...] ctrl = simulate("examples/DummyFMUs/Controller.fmu");
  Controller control = new Controller(ctrl, this, this.nextRoom);
  if this.ctrl != null then destroy(this.ctrl); end
  this.ctrl = control;
  this.nextRoom.ctrl = control;
  if this.nextRoom.nextRoom != null then
    this.nextRoom.nextRoom.rebuildCtrl();
  end
end
```

**Fig. 11.** Rebuilding the controller structure.

that the knowledge base must is consistent. One can also use queries, by giving special SPARQL queries or OWL classes that must be empty or return an empty answer set. The approach sketched above can be seen as a variation of this idea: the query detecting structural drift formulates the constraint that the DT and the PT are consistent with each other. Such queries can also be used without a following repair.

Alternatively, one can perform data validation using SHACL, which is a more lightweight approach as it does not involve reasoning. For example, to validate the DTI we can formulate that the room that is stored in the `House.firstRoom` field is indeed the first one from the left, as the following SHACL shape.

```
schema:FirstShape a sh:NodeShape ;
  sh:targetClass prog:House ;
  sh:property [
    sh:path
      (prog:House_firstRoom prog:Room_wallLeft prog:Wall_areaLeft);
    sh:class prog:Outside;
  ].
```

It expresses that for every node that is of `prog:House` class, following the path `prog:House_firstRoom prog:Room_wallLeft prog:Wall_areaLeft` ends in an object of type `prog:Outside` . I.e., the area to the left of the left wall of the first room must be outside.

*Removal of Assets.* To remove objects that are no longer part of the asset model, we run a similar query as before, but must consider that an asset that is removed from the physical system is not removed from the KB, but instead marked as removed. An example for such a query, which directly returns all the `Room` instances to be removed is the following. We refrain from giving the repair methods, which are analogous to adding assets.

```
SELECT ?y WHERE { ?x asset:Room_id ?id;
                     a asset:removed.
                  ?y a prog:Room; prog:Room_id ?id. }
```

## 6 Conclusion

We have presented an approach to reconfiguring digital twin infrastructure according to structural changes. The approach integrates a knowledge base with a digital twin infrastructure. The knowledge base includes an asset model, which formalises our knowledge of the physical twin, with a similar representation of the runtime state of the digital twin. We provide a proof of concept implementation of the approach in `SMOL`, a programming language which allows the runtime state of programs to be lifted into a knowledge base and queried from within the running program (so-called semantic reflection). We implement the DTI as a `SMOL` program and view the physical twin through an asset model. Both DTI and asset models are integrated into a knowledge base, so that the DTI can perform semantic reflection and perform queries on itself and the asset model to detect discrepancies and guide the reconfiguration. The very same integrated knowledge base is also used to validate domain specific constraints on the DTI and the relation of the DTI to asset model.

Our proof of concept implementation in `SMOL` has assumed a one-to-one relation between the components of the asset model and those of the simulation system. In future work, we will explore other relations between the structure of the asset and the structure of the digital twin. Furthermore, we aim to automatically generate the digital twin infrastructure from a top-down asset model, including automatic detection of structural drift and repair, using the more advanced RDF loading mechanism recently developed for `SMOL` [20]. Our work so far does not incorporate data streams from the asset into the knowledge base. We expect that this integration can be handled similar to the semantic lifting of the runtime state into the knowledge base's static structure of the digital twin, but this remains to be done. Whereas the knowledge base is well suited to store and query information, solving constraints is not directly supported (e.g., for parameter optimisation). We believe that this apparent limitation of the approach can be naturally overcome by using the knowledge base to collect constraints, to be solved by an external solver.

## References

1. R. Anderl, S. Haag, K. Schützer, and E. Zancul. Digital twin technology – An approach for Industrie 4.0 vertical and horizontal lifecycle integration. *it - Information Technology*, 60(3):125–132, June 2018.
2. A. Banerjee, R. Dalal, S. Mittal, and K. P. Joshi. Generating digital twin models using knowledge graphs for industrial production lines. In *Proc. Web Science Conf. (WebSci'17)*, page 425–430. ACM, 2017.
3. J. Bickford, D. L. Van Bossuyt, P. Beery, and A. Pollman. Operationalizing digital twins through model-based systems engineering methods. *Systems Engineering*, 23(6):724–750, 2020.
4. T. Blochwitz, M. Otter, J. Åkesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models. In *Modelica Conf.*, pages 173–184. The Modelica Association, 2012.
5. M. Bolpagni. Building Information Modelling and Information Management. In M. Bolpagni, R. Gavina, and D. Ribeiro, editors, *Industry 4.0 for the Built Environment: Methodologies, Technologies and Skills*, Structural Integrity, pages 29–54. Springer, 2022.
6. D. B. Cameron, A. Waaler, and T. M. Komulainen. Oil and Gas digital twins after twenty years. How can they be made sustainable, maintainable and useful? In *Proc. 59th Conf. on Simulation and Modelling (SIMS 59)*, pages 9–16. Linköping University Electronic Press, 2018.
7. P. Delgoshaei, M. A. Austin, and D. A. Veronica. A Semantic Platform Infrastructure for Requirements Traceability and System Assessment. In *Ninth Intl. Conf. on Systems (ICONS 2014)*. IARIA, Feb. 2014.
8. H. Feng, C. Gomes, C. Thule, K. Lausdahl, A. Iosifidis, and P. G. Larsen. Introduction to digital twin engineering. In C. R. Martin, M. J. Blas, and A. Inostrosa-Psijas, editors, *Annual Modeling and Simulation Conference, ANNSIM 2021, Virtual Event / Fairfax, VA, USA, July 19-22, 2021*, pages 1–12. IEEE, 2021.
9. E. Fjøsna and A. Waaler. READI Information modelling framework (IMF). Asset Information Modelling Framework. Technical report, READI Joint Industry Project, Mar. 2021.
10. A. Fraga, J. Llorens, L. Alonso, and J. M. Fuentes. Ontology-Assisted Systems Engineering Process with Focus in the Requirements Engineering Process. In F. Boulanger, D. Krob, G. Morel, and J.-C. Roussel, editors, *Complex Systems Design & Management*, pages 149–161. Springer, 2015.

11. B. Glimm and M. Krötzsch. SPARQL beyond subgraph matching. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *Proc. 9th Intl. Semantic Web Conf. (ISWC 2010)*, volume 6496 of *Lecture Notes in Computer Science*, pages 241–256. Springer, 2010.
12. B. Glimm and C. Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Recommendation, 2013. Available at `http://www.w3.org/TR/sparql11-entailment/`.
13. C. Gomes, L. Lúcio, and H. Vangheluwe. Semantics of co-simulation algorithms with simulator contracts. In *MoDELS (Companion)*, pages 784–789. IEEE, 2019.
14. C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe. Co-simulation: A survey. *ACM Comput. Surv.*, 51(3):49:1–49:33, 2018.
15. M. Grieves and J. Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In F.-J. Kahlen, S. Flumerfelt, and A. Alves, editors, *Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches*, pages 85–113. Springer, 2017.
16. J. Heaton and A. K. Parlikad. Asset information model to support the adoption of a digital twin: West cambridge case study. *IFAC-PapersOnLine*, 53(3):366–371, 2020. 4th IFAC Workshop on Advanced Maintenance Engineering, Services and Technologies - AMEST 2020.
17. IEC TC3. IEC 81346-1 Structuring principles and reference designations - Part 1 Basic rules. International Standard IEC 81346-1 Ed. 1, IEC, July 2009.
18. IOGP JIP 36. CFIHOS Standards. `https://www.jip36-cfihos.org/cfihos-standards/`. Accessed: 2021-12-12.
19. E. Kamburjan and E. B. Johnsen. Knowledge structures over simulation units. In *Proc. SCS Annual Modeling and Simulation Conf. (ANNSIM 2022)*, 2022. In press.
20. E. Kamburjan, V. N. Klungre, and M. Giese. Never mind the semantic gap: Modular, lazy and safe loading of RDF data. In *Proc. 19th Intl. Conf. on the Semantic Web (ESWC 2022)*, volume 13261 of *Lecture Notes in Computer Science*, pages 200–216. Springer, 2022.
21. E. Kamburjan, V. N. Klungre, R. Schlatte, E. B. Johnsen, and M. Giese. Programming and debugging with semantically lifted states. In R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski, and M. Alam, editors, *Proc. 18th Intl. Conf. on the Semantic Web (ESWC 2021)*, volume 12731 of *Lecture Notes in Computer Science*, pages 126–142. Springer, 2021.
22. E. Kharlamov, F. Martín-Recuerda, B. Perry, D. Cameron, R. Fjellheim, and A. Waaler. Towards semantically enhanced digital twins. In *IEEE BigData*, pages 4189–4193. IEEE, 2018.
23. E. V. Kostylev and B. C. Grau. On the semantics of SPARQL queries with optional matching under entailment regimes. In *ISWC*, pages 374–389, 2014.
24. D. Leal. ISO 15926 "Life Cycle Data for Process Plant": an Overview. *Oil & Gas Science and Technology*, 60(4):629–637, July 2005.
25. P. Lietaert, B. Meyers, J. V. Noten, J. Sips, and K. Gadeyne. Knowledge graphs in digital twins for AI in production. In A. Dolgui, A. Bernard, D. Lemoine, G. von Cieminski, and D. Romero, editors, *Proc. IFIP WG 5.7 Intl. Conf. on Advances in Production Management Systems. Artificial Intelligence for Sustainable and Resilient Production Systems (APMS 2021)*, volume 630 of *IFIP Advances in Information and Communication Technology*, pages 249–257. Springer, 2021.
26. R. Mehmandarov, A. Waaler, D. Cameron, R. Fjellheim, and T. B. Pettersen. A semantic approach to identifier management in engineering systems. In *Proc. Intl. Conf. on Big Data (Big Data)*, pages 4613–4616. IEEE, 2021.

27. C. Nigischer, S. Bougain, R. Riegler, H. P. Stanek, and M. Grafinger. Multi-domain simulation utilizing SysML: state of the art and future perspectives. *Procedia CIRP*, 100:319–324, Jan. 2021.
28. B. J. Oakes, B. Meyers, D. Janssens, and H. Vangheluwe. Structuring and accessing knowledge for historical and streaming digital twins. In I. Tiddi, M. Maleshkova, T. Pellegrini, and V. de Boer, editors, *Joint Proc. of the Semantics co-located events: Poster&Demo track and Workshop on Ontology-Driven Conceptual Modelling of Digital Twins*, volume 2941 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.
29. A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semant.*, 10:133–173, 2008.
30. E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C Recommendation, 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.
31. READI. Reference Designation System for Oil and Gas – READI, 2020.
32. M. Rotondi, A. Cominelli, C. Di Giorgio, R. Rossi, E. Vignati, and B. Carati. The benefits of integrated asset modelling: lessons learned from field cases. In *Europec/EAGE Conf. and Exhibition*. OnePetro, 2008.
33. M. G. Skjæveland, M. Giese, D. Hovland, E. H. Lian, and A. Waaler. Engineering ontology-based access to real-world data sources. *J. Web Semant.*, 33:112–140, 2015.
34. Ø. R. Smogeli, K. B. Ludvigsen, L. Jamt, B. Vik, H. Nordahl, L. T. Kyllingstad, K. K. Yum, and H. Zhang. Open simulation platform – an open-source project for maritime system co-simulation. In *COMPIT*. Technische Universität Hamburg-Harburg, 2020.
35. H. Sohier, P. Lamothe, S. Guermazi, M. Yagoubi, P. Menegazzi, and A. Maddaloni. Improving simulation specification with MBSE for better simulation validation and reuse. *Systems Engineering*, 24(6):425–438, 2021.
36. F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee. Digital twin in industry: State-of-the-art. *IEEE Trans. Ind. Informatics*, 15(4):2405–2415, 2019.
37. W3C, OWL Working Group. Web ontology language. `https://www.w3.org/OWL`.
38. W3C, RDF Working Group. Resource description framework. `https://www.w3.org/RDF`.
39. W3C, SHACL Working Group. Shapes constraint language. `https://www.w3.org/TR/shacl/`.
40. M. Waszak, A. N. Lam, V. Hoffmann, B. Elvesæter, M. F. Mogos, and D. Roman. Let the asset decide: Digital twins with knowledge graphs. In *19th IEEE Intl. Conf. on Software Architecture (ICSA 2022)*. IEEE, 2022.
41. K. Wei, J. Z. Sun, and R. J. Liu. A review of asset administration shell. In *2019 IEEE Intl. Conf. on Industrial Engineering and Engineering Management (IEEM)*, pages 1460–1465, 2019.
42. M. Wiedau, L. von Wedel, H. Temmen, R. Welke, and N. Papakonstantinou. ENPRO Data Integration: Extending DEXPI Towards the Asset Lifecycle. *Chemie Ingenieur Technik*, 91(3):240–255, 2019.
43. H. Yan, J. Yang, and J. Wan. KnowIME: A system to construct a knowledge graph for intelligent manufacturing equipment. *IEEE Access*, 8:41805–41813, 2020.
44. J. Zhang, H. Luo, and J. Xu. Towards fully BIM-enabled building automation and robotics: A perspective of lifecycle information flow. *Computers in Industry*, 135:103570, Feb. 2022.
45. B. Zhou, Y. Svetashova, A. Gusmao, A. Soylu, G. Cheng, R. Mikut, A. Waaler, and E. Kharlamov. SemML: Facilitating development of ML models for condition monitoring with semantics. *J. Web Semant.*, 71:100664, 2021.