Semantic Reflection and Digital Twins: A Comprehensive Overview

Eduard Kamburjan, Andrea Pferscher, Rudolf Schlatte, Riccardo Sieve, Silvia Lizeth Tapia Tarifa, and Einar Broch Johnsen

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

Abstract. Semantic reflection combines reflection in programming languages with semantic technologies for knowledge representation. It enables a program to represent and query its own runtime state as a knowledge graph. The knowledge graphs reflecting program states can be combined with domain knowledge which allows queries about a program to be made in terms of a given domain vocabulary, as well as with external graph data. Both extensions of the knowledge graph reflecting the runtime state are useful for digital twins. In this paper, we discuss the basic concepts of semantic reflection, its applications for digital twins, and its connections to formal methods.

1 Introduction

Digital twins propose a model-based approach to cyber-physical systems, in which a physical system (the "physical twin") is connected with a digital system (the "digital twin"). The digital twin contains models of the physical twin and connects these models with the physical twin to (a) update the models with live data and (b) enable control over the physical twin using different kinds of model-based analyses. While the exact definition of a digital twin is elusive (e.g., [11, 12, 44]) one important perspective in engineering is that the life-time of the digital twin goes beyond the operational phase to reflect the lifecycle of the physical twin, already starting with the requirement elicitation phase. Thus, documents and models that are collected through the different phases of the lifecycle become part of the digital twin, which, together with tools to support design, construction, and operation, are often referred to as a *digital thread* [50].

Developing such digital twins, with their inherent multidisciplinarity and their multitude of tools and data formats, is a major challenge for both software developers and system architects. One way to deal with interdisciplinarity, is through the formal representation of domain knowledge in knowledge graphs. While ontologies, knowledge graphs [25] and related semantic technologies [24] have been identified as a key technology for digital twins [41, 61], their use in turn poses additional challenges to the software developer [23].

In this paper, we discuss our work on *semantic lifting* [34] in the context of digital twins. Semantic lifting is the process of (1) serializing a digital entity as

a knowledge graph, (2) connecting this knowledge graph to domain knowledge necessary to operate or interpret the digital entity, and (3) accessing this combined knowledge graph from the original digital entity. The last step is referred to as *semantic reflection*. Specifically, we consider the use of *knowledge graphs* to

1. model the architecture and configurations of digital twins during development as well as operation, and
2. enable digital twins to access their own runtime configuration in order to adapt their future behavior.

Together, these properties enable *procedural reflection* [43, 58] through semantic technologies, such as ontologies and graph queries. Semantic reflection does not only enhance digital twins, but can also be used to provide tool support and integrate with formal methods. So far, semantic reflection has been used to adapt to *structural changes* in the digital twin [35, 40], check structural correctness [30, 32, 35], interpretation of software architectures [16] and to enable new techniques for programming [33] with semantic graph data. In this paper, we show the connections and interactions between these applications.

Paper outline. We provide an overview of our work on semantic reflection (Section 2), with a focus on digital twins (Section 3). We then discuss software development and analysis for digital twins (Section 4), and some perspectives on integrating formal methods into digital twins by extending our techniques (Section 5). Examples and content in this paper builds on previous publications, which also detail the corresponding related work; the contribution of this paper is the comprehensive overview.

2 From Semantic Lifting to Semantic Reflection

Let us illustrate the basic idea of semantic lifting of programs [34] and how semantic lifting can be used to integrate domain knowledge in programs through reflection. Throughout the paper we use as a running example a simple program of a smart house [35], written in the *Semantic Micro Object Language* (SMOL).[1] This example will gradually become a digital twin.

Consider a smart house that consists of a sequence of rooms, each modeled by an instance of a class `Room` with an identifier and a target temperature for heating. The code in Figure 1 shows an excerpt of the corresponding SMOL code. The main block instantiates a house with two rooms; we will detail its control method later.

2.1 Semantic Lifting

Semantic lifting is a procedure that generates a knowledge graph from a program state. The generated knowledge graph includes a representation of the full

[1] https://www.smolang.org

SMOL

```
class Wall(Room left, Room right) end
class Room(Wall left, Wall right, Int id, Int target)
  Unit control() ... end
end

main
  Wall w1 = new Wall(null, null); Wall w2 = new Wall(null, null);
  Wall w3 = new Wall(null, null);
  Room r1 = new Room(w1, w2, 1, 18);
  Room r2 = new Room(w2, w3, 2, 19);
  w1.right = r1; w2.right = r2; w2.left = r1; w3.left = r2;
end
```

Fig. 1. A SMOL model for a digital twin of a smart house.

RDF

```
run:obj1 a prog:Wall; prog:Wall_right run:obj4.
run:obj2 a prog:Wall; prog:Wall_right run:obj5;
                      prog:Wall_left run:obj4.
run:obj3 a prog:Wall; prog:Wall_left run:obj5.
run:obj4 a prog:Room; prog:Room_left run:obj1;
                      prog:Room_right run:obj2;
                      prog:Room_target 18;
                      prog:Room_id 1.
run:obj5 a prog:Room; prog:Room_left run:obj2;
                      prog:Room_right run:obj3;
                      prog:Room_target 19;
                      prog:Room_id 2.
```

Fig. 2. A lifted final state of the program in Figure 1 (excerpt).

runtime state — object, variables, heap memory, call stack — but also the static information of the program, in particular the class table and the abstract syntax tree of the program.

To illustrate, Figure 2 contains an excerpt of the lifting of the smart house program shown in Figure 1, at the end of its execution, with a knowledge graph generated from the objects and parts of the class table, represented in RDF.[2] As we can see, the architectural structure of the digital twin is clearly expressed in the lifted program state — in particular, the configuration of the smart house which shows how the two rooms are connected, can be retrieved from the graph. In anticipation of the digital twin this program will become, we can already see that the lifted graph expresses the architectural structure of the house *as it is modeled at a given moment.*

[2] Resource Description Framework, https://www.w3.org/RDF/

OWL

```
Class: prog:Room SubClassOf: SMOLClass
Class: prog:Wall SubClassOf: SMOLClass
ObjectProperty: prog:Room_left Domain: prog:Room
                               Range: prog:Wall or {smol:null}
ObjectProperty: prog:Room_right Domain: prog:Room
                                Range: prog:Wall or {smol:null}
ObjectProperty: prog:Wall_left Domain: prog:Wall
                               Range: prog:Room or {smol:null}
ObjectProperty: prog:Wall_Right Domain: prog:Wall
                                Range: prog:Room or {smol:null}
DataProperty: prog:Room_id Domain: prog:Room
                           Range: xsd:int
                           Characteristics: functional
DataProperty: prog:Room_target Domain: prog:Room
                               Range: xsd:int
                               Characteristics: functional

ObjectProperty: leftOf
   EquivalentTo: inverse(prog:Wall:right) o prog:Wall_left
   Domain: prog:Room
   Range: prog:Room
```

Fig. 3. Axioms for the lifted state in Figure 2. Observe that only the last axiom is part of the external knowledge graph.

Merely generating the knowledge graph, which is essentially serialization, is not using the potential of knowledge graphs — but by connecting the generated knowledge graph with ontologies, we can add domain knowledge to the program state to investigate it and reason over it. We say that the lifted stated is *enriched* with information from an external knowledge graph.

For example, consider the OWL[3] axioms in Figure 3. They express the domain and range of different relations in the knowledge graph, and declare some concepts to be OWL classes. The final axiom captures the spatial relation between two rooms. Some axioms are already part of the generated knowledge graph (the axioms concerned with range and domain, as well as the OWL classes related to the lifting of the runtime state). Remark that the knowledge graph that the lifted runtime state connects to need not necessarily be an ontology, but can also contain concrete data; we shall return to this point in later sections.

Using the generated knowledge graph and the domain ontology, we can now query the semantically lifted runtime state *in terms of the domain*. The SPARQL[4] query in Figure 4, for example, asks for all rooms that are left of another (known) room. This information must be derived using the above axioms.

[3] Web Ontology Language, https://www.w3.org/OWL/
[4] SPARQL RDF Query Language, https://www.w3.org/TR/sparql11-overview/

SPARQL

```
SELECT ?obj WHERE { ?obj leftOf ?obj2.}
```

Fig. 4. A query on the lifted state from Figure 2, using the axioms from Figure 3. The result of the query is the node `run:obj4`.

SMOL

```
... // Classes as in Figure 1
main
  ... // Initialization as in Figure 1
  List<Room> rooms =
    access("SELECT ?r WHERE {?r leftOf ?r2. ?r prog:Room_id ?id1.
                             ?r2 prog:Room_id ?id2.
                             FILTER(?id1 = ?id2) }");

   //every result is a room that has the same id as its neighbor
   //the following should print 'false'
   print(rooms != nil);
end
```

Fig. 5. Semantic reflection for checks in terms of the domain.

Observe that this query abstracts from internal data structures used to organize rooms in the program.

2.2 Semantic Reflection

Building on semantic lifting, *semantic reflection* is the process of accessing the generated and enriched knowledge graph of a program state from inside the very same program (introspection) to influence its future behavior (intercession).

For this purpose, we need to execute graph queries from within the program. The queries can either return a Boolean value (denoting whether there is a result for the query) or are restricted to return representable results (for example, through a static type system [33]). A result, for the purposes of our example here, is *representable* if it is an RDF node corresponding to a SMOL object. The result of a reflective query then becomes a list of SMOL objects. Consider the variant of our smart house in Figure 5, which queries itself to check whether the identifiers differ for all rooms such that one room is left of the other. The **access** statement executes a query on the knowledge graph with the lifted state. The result is assigned to the list `rooms`, which should be the empty list (denoted `nil`).

Semantic reflection in practice. A typical use case for semantic reflection is the development of domain-specific simulators. For example, the geological simulator of Qu *et al.* [55] directly integrates the GeoFault [56] and GeoCore [15]

ontologies into the simulation of geological processes, thus removing most redundancy between the two modeling formalisms. By combining these ontologies into a simulator, geological deliberations that take days, can be supported with simulations that return *qualitative* results within minutes.

3 Foundations of Semantically Reflective Digital Twins

The previous section showed how programs can use domain knowledge expressed in a knowledge graph either through reflection or through access to external knowledge. We now consider how these techniques can be used to program digital twins. The main insight here is that the knowledge graph that enriches the lifted state can contain information about the physical twin, such as its architectural structure, requirements and other asset information.

3.1 Using Asset Models for Architectural Self-Adaptation

An *asset information model* is a digital description of a physical or planned asset to facilitate its design, development and operation.[5] We consider the asset information model to be a part of the digital thread related to an asset. The digital thread additionally contains information that describes the context of the asset as well as, e.g., operational logs.

An asset information model describes the architectural structure of a physical twin, including its components and their connections. We are specifically interested in this structure of the physical twin: it changes throughout the lifetime of the physical twin, and the digital twin must adapt *its own architectural structure* to reflect these changes.

The digital twin and the physical twin have different architectural structures, as they have different components and represent different abstraction levels, but there is still a relation between these structures. For example, in our running example, the digital twin will have one object that represents each room of the smart house. This `Room` object in the digital twin should evolve in sync with the corresponding room in the physical twin, which it can realize by monitoring the room in the physical twin. This monitoring is typically related to *requirements* connected to the physical twin (e.g., the targeted temperature for the room). If the physical twin changes, it may be necessary to adapt the digital twin, but not all changes require an adaptation. For example, adding a room to the smart house or adding additional requirements from operations would make an adaptation of the digital twin necessary, but turning on the light in a room would not.

The relationship between the physical twin and the digital twin can be seen as a *structural consistency relation*, expressing that the architectural structure of the digital twin is consistent with that of the physical twin. Semantic lifting

[5] Asset information models generalize the building information model (BIM) [26] to systems engineering (e.g., [14]).

can be used to formalize consistency between the architectural structures in a uniform way, despite the potentially different nature of the two architectures: As both the asset information model and the digital twin (via the semantic lifting) are part of the same knowledge graph, we can define structural consistency using queries. This has several advantages:

- The formalization of consistency is decoupled from the exact nature of the digital twin implementation. Changing from, for example, one programming language to another for the digital twin, does not require to reformulate structural consistency.
- Structural consistency can be expressed using standard semantic web technologies (concretely, SPARQL in our case), for which advanced pragmatics are developed to ease their use. In particular, structural consistency can be expressed in a formalism specifically design for *structural modeling*.
- By querying the knowledge graph, we can easily integrate consistency and self-adaptation.

Technically, we define consistency between architectural structures using *defect queries* [16]. A defect query defines *an inconsistency* and returns all instances of the inconsistency in the graph. For example, given a program, an asset information model and a set of defect queries, we consider the program and the asset information model to be structurally consistent if all the defect queries return an empty result set. Note that logical reasoning using the ontology may be needed to derive the answers (i.e., a so-called entailment regime from the query engine).

Strategies for *architectural self-adaptation* [6] can be easily expressed by means of defect queries [35], using, e.g., the MAPE-K self-adaptation loop (see, e.g., [9]). In the MAPE-K framework, a managing subsystem monitors a managed subsystem, analyzes found inconsistencies, plans their repair and executes that plan.

For architectural self-adaptation of a digital twin, the managing subsystem is an additional component in the digital twin, while the managed subsystem contains the other, original, digital twin components. As every query returns an inconsistency, the set of defect queries act as the monitoring components at the meta-level, monitoring the architectural structure of the digital twin. The analysis component than analyses query results to detect the source of the inconsistency. This can be either a more detailed query, or just some computation without referring to the knowledge graph. Planning and execution can be tailored to the defect queries and the inconsistencies that they model.

Let us now reconsider the running example of the smart house. The code in Figure 6 illustrates how semantic reflection can be used by the digital twin to self-adapt to changes in the architectural structure of the physical twin. First, we retrieve all room identifiers in the asset model that do not have a digital twin object modeling them (Lines 6–9). This means that these rooms have been added to the physical twin recently. The **construct** creates a new object instance from retrieved data values. This general form of a defect query generalizes for this kind of constraints [35]. The next query analyses the defect and retrieves the wall neighboring the new room (Lines 11–16). One of these walls must be

```
1 ... //Classes as in Figure 1
2 class RoomAsrt(Int roomID) end
3 class RoomNeigh(Int leftWallId, Int rightWallId) end
4 class MAPE()
5  Unit selfAdapt()
6   List<RoomAsrt> newRooms =  // (1) Monitoring with defect query
7    construct("SELECT ?roomID
8                  { ?x asset:Room_id ?roomID.
9                    FILTER NOT EXISTS {?y prog:Room_id ?roomID}}");
10  foreach roomAsrt in newRooms do
11   List<RoomNeigh> neighbors = //  (2) Analysis
12   construct("SELECT ?leftWallId ?rightWallId
13                  { ?x asset:Room_id %1;
14                    ?x asset:left  [asset:Wall_od ?leftWallId];
15                    ?x asset:right [asset:Wall_od ?rightWallId].",
16              roomAsrt.roomId);
17    // determine whether one of the walls is known, plan and
18    // execute the creation of a new room in the digital twin
19  end
```

Fig. 6. Architectural self-adaptation in SMOL.

known if we assume that rooms are added one by one. SMOL allows values to be injected in the queries using % and these queries are specific to one found defect.

3.2 Beyond the Monolith

Digital twins are not in general monolithic programs, but consist of multiple models and data sources, some of which are typically (black-box) simulators. It is unrealistic to assume that all components of a digital twin are expressed in the same programming language, and numerous architectures and platforms for digital twins have been proposed [46], including one by Margaria *et al.* [7]. We now discuss how semantic lifting and reflection can be applied in such architectures, with a focus on the following approaches:

- **Semantically reflected orchestrators.** Orchestrators are central components of the digital twins and integrate the connection to other components. In semantically reflected orchestrators, only the orchestrator, connections and interfaces of other components are lifted into the knowledge graph.
- **Semantically reflected architectures** define their lifting not in terms of the programming language, but in terms of the components of the architecture. The system is semantically lifted (and can semantically reflected) based on a lifting of all components *with respect to the architecture*.

We now provide more detail for these two approaches.

Semantically reflected orchestrators. We have investigated the semantically reflected orchestrators approach in SMOL by integrating simulation units into the language and considering them as special objects in the language [32]. These objects can realize either a simulator or a connection to the physical twin, as their interface consists of only three methods: reading an object port, writing an object port and advancing the local (simulation) time of the object.

More concretely, we integrate functional mock-up units (FMUs) [5] into SMOL. An FMU is a wrapper defined by the functional mock-up interface that offers the above operations. Each FMU comes with a model description, which is part of the semantic lifting, that describes its ports, name of the model and internal details that must be exposed for co-simulation. However, it does not expose the implementation of the simulation — indeed it must not implement a simulation at all, but can connect to other temporal data sources.[6] Continuing with our running example, the code in Figure 7 places two FMUs in each room: one FMU to simulate the behavior of the room, the other to connect to the sensors of the physical twin. The shown code uses semantic lifting to check that the correct FMUs are used; this is done by a query comparing the model name of the simulation FMU with information in the asset model. SMOL also supports a similar connection for InfluxDB.[7] A semantically reflected orchestrator, embedded into architecture that is not lifted, has been successfully demonstrated on an extensible digital twin architecture for a greenhouse [40].

Semantically reflected architectures. We have investigated semantically reflected architectures in a service-oriented setting for a digital twin of robotic arms [16]. The digital twin is defined by a software architecture, expressed in UML, with more than 10 different software components: databases, services, simulators and physical twin endpoints. In this architecture, there is a *lifting service* that generates a knowledge graph for the current instance of the software implementing the architecture. This means that each component implements an interface to lift itself. Instead of relying on a generic lifting function provided by the language (which is the case for SMOL), the semantic lifting must here be implemented by hand, in terms of the vocabulary defined by the architecture (such as the components and attribute names). Currently, there is on-going effort to implement semantic lifting as a service on a Digital-Twin-as-a-Service (DTaaS) platform [59]. Additional information and specification can be annotated to the architecture and then be used in the semantic lifting.

There are numerous digital twin platforms and architectures [46], and several proposals that combine digital twins with ontologies and knowledge graphs [41, 57,61]. However, no general methodology has been developed so far to determine whether a lifted architecture or a lifted orchestrator is best suited for a given system; we consider this challenge the next step in research on semantically reflected digital twins.

[6] See, e.g., the RabbitMQ FMU https://into-cps-rabbitmq-fmu.readthedocs.io/en/latest/overview.html.

[7] https://www.influxdata.com/products/influxdb-overview/

SMOL

```
class Room(Wall left, Wall right, Int id, Int target,
           FMU[in Boolean switch, out Double value] room,
           FMU[in Boolean switch, out Double value] model)
  Unit control()
     if /* check on room.value and model.value */ then //start heating
        room.switch = True; model.switch = True;
     end
     room.tick(1.0); model.tick(1.0); //advance time
  end
end
class ConsistencyManager()
 List<Room> getMisconfigured() //should return nil
  return
   access("SELECT ?obj {?obj prog:Room_room ?fmu.
                        ?fmu fmu:name ?fName.
                        FILTER(?fName != 'connectionFMU')}")
   ++ //syntactic sugar for list concatenation
   access("SELECT ?obj {?obj prog:Room_model ?fmu.
                        ?fmu fmu:name ?fName.
                        FILTER(?fName != 'simulationFMU')}");
 end
end
```

Fig. 7. SMOL code with integrated FMUs.

4 Analysis of Semantically Reflected Programs

At the core of semantic reflection is the ability to program with semantic graph data, a challenging task sometimes referred to as the next big topic in the semantic web [23]. It requires not only to coordinate different tools (reasoners, query endpoints, databases) and formalisms (graph shapes, ontologies, queries), but also to integrate their conceptual class models with object-oriented class models. While these two class models seem similar, they are in fact fundamentally different. Not only technical terms, but also in their very purpose: an object-oriented model is concerned with structuring data and behavior, a conceptual class model (i.e., an ontology) is concerned with the modeling of domain knowledge.[8]

The semantic gap [2] between object-oriented models and knowledge representation can be addressed using semantic lifting. As semantic lifting expresses the program using an ontology, the semantic lifting process can be used to analyze the query that loads data into the program; the answer to the query must

[8] Remark that the Scandinavian school of object-oriented design, stemming from Simula [10], places as much weight on modeling and simulation as on code reuse in object-oriented systems [48, 49].

be contained in the part of the knowledge graph defined by the lifting. In fact, we have exploited this feature in the above examples.

Consider the following statement:

SMOL
```
List<Room> r = access("SELECT ?obj {?obj leftOf ?obj2}"); //query Q
```

To check whether this query is type-safe, one must reason about the ontology $\mathcal{O}$ (from Figure 3) used with the program: Is the query `Q` always returning a collection of `Room` objects *on any knowledge graph that adheres to* $\mathcal{O}$? This can be answered using *query containment* [8,54]. We have shown that SMOL is type-safe for queries that can be be reduced to concept subsumption [37]. Namely, if the query `Q` is contained in the query

SPARQL
```
SELECT ?obj WHERE { ?obj a prog:Room. }
```

using $\mathcal{O}$, then no runtime error can be triggered by a query result that cannot be represented as a `Room` object. It is easy to see that this is indeed the case, from the axioms defining the `leftOf` relation. This idea can be further generalized to define SMOL classes not in terms of a ontology concept, but in terms of the queries that retrieve them [33], which introduces behavioral subtyping [47] for program classes interacting with ontological knowledge bases.

Formal methods with semantic lifting. Semantic reflection opens for interesting questions in formal methods. We briefly describe two directions that we have started to explore. new ways of doing formal met First, semantic reflection can be used for runtime enforcement [29]. Using hypothetical execution, one can use the lifted state of a program *after* a potential step, to check for consistency with the domain knowledge. If a step would result in a inconsistent knowledge graph, then the runtime forces another step to be taken, thus ensuring that domain knowledge is adhered to during execution, even without an explicit, reflective access. Second, semantic lifting can be used for software verification, foe example by integration in Hoare logics [31] for deductive verification. While semantic reflection is so far out of reach, it enables the use of ontologies as a specification language (together with program expressions), thus opening a new line of attack to the old problem of the specification bottleneck [3].

5 Discussion: Formal Methods in Digital Twins

We now consider connections between formal methods and digital twins (a broader discussion may be found in the paper from the FMDT workshop [36]). First, observe that the semantic lifting mechanism discussed in this paper has connections to formal methods, analysis and verification for digital twins. The defect queries, discussed in Section 3.1, can be exploited as a mechanism for self-adaptation, in particular for structural changes. Alternatively, they can also be seen as correctness conditions that need to be monitored at runtime [30].

The orchestration and coupling of simulation units, discussed in Section 3.2, can be subjected to runtime monitoring to ensure the correct use of co-simulation master algorithms [18] and in changing scenarios [21]. While processes, traces and temporal properties may be challenging to model generically using ontologies, due to their dependence on a concrete application [22], this shows that temporal properties can still be used to specify correctness properties for digital twins using semantic technologies.

In addition to the use of semantic lifting and semantic reflection for verification, we briefly mention some other topics in formal methods that are relevant for digital twins.

Hybrid systems verification. So far in this paper, we have discussed a largely language-based approach to connect and analyze code and simulators as discrete structures. However, this impedes the formal analysis of continuous behavior and sensor streams. There are numerous languages for hybrid systems modeling, and we envision that an integration with a structured, object-oriented formalism to hybrid systems, such as hybrid active objects [28, 38, 39] or hybrid actors [27] can result in a holistic formal method for verification of digital twins.

Automata learning. Automata learning [17] is a technique to automatically generate behavioral models of black-box systems. Learning algorithms have been developed to learn models describing behavioral aspects such as timed or stochastic behavior. Margaria and Schieweck [51] propose automata learning as a technique to automatically create a digital twin of cyber-physical systems. Similarly, Pferscher *et al.* [53] showed that this can be done for security-critical communication protocols. The learned models that enable further analyses, e.g., model-checking or model-based testing. Similar to our proposed technique, the learned model enables insights into the physical twin. Wallner [60] has shown that automata learning can also be used to analyze the digital twin. By applying automata learning also *to* the digital twin, the behavioral difference between the digital twin and the physical twin can be analyzed.

Runtime enforcement and extra-functional properties. The value proposition of digital twins is largely based on its ability to incorporate live data with model-based analysis to aid in decision-making. In many of these cases, this data may be *sensitive*. Data lakes [13, 20] can be used to collect and organize this data space in a decentralized manner. The access to data from different sources needs to be carefully managed through fine-grained access control policies; for example, Margaria *et al.* proposed an algorithm for role-based access control in digital twins [7]. Furthermore, these data access policies can be connected with data privacy, expressing, e.g., GDPR compliance, and connected with language-based methods to enforce data privacy consent (e.g., [1, 42]). It would be interesting to explore whether semantic reflection could be used for runtime enforcement of dynamic privacy policies that changes at runtime, such as data privacy consent as required by GDPR, for digital twins that need to access personal personal data. Going beyond the use of personal data for analysis in the digital twin, such

runtime enforcement could potentially be directly related to the digital thread, thereby regulating data access throughout the digital twin lifecycle [7, 45].

Concerns on responsible decision making. Responsible decision support using digital twins to explore and compare hypothetical scenarios, will increasingly require solutions that take into consideration transparency, explainability, human-centric values, and the law [52]. It would be interesting to investigate whether semantic reflection, as outlined in this paper, could be used to address responsible decision-making and decision support, e.g., using knowledge graphs and runtime analyses as a means for argumentation in the decision-making process and to capture human-centric values and legal regulations; in this direction, Gruber discusses how collective intelligence can be captured via collective knowledge systems [19], while Becu *et al.* [4] explores how simulation approaches can be used for participatory decision-making, where all stakeholders' views are taken into consideration.

6 Conclusion

Semantic reflection is a language-based approach to programming with graph data and ontologies, that has proven useful in the context of digital twins. In particular, semantic reflection can be used to connect software to the digital thread and to asset models, thereby supporting a lifecycle perspective on the digital twin. In this paper, we have provided an overview of our work in this area. We further suggest how semantic reflection can be used to draw interesting connections between three different areas of computer science: programming languages, formal methods and knowledge representations. So far, the work on formal methods has been investigated less than the connection with digital twins, and we hope that the comprehensive overview and perspective given here can help foster further research on this topic.

Acknowledgments The authors would like to thank Tiziana Margaria for many interesting discussions on digital twins. We are further grateful to all contributors to SMOL and its development. This work was partly funded by the EU project SM4RTENANCE (grant no. 101123423) and the Research Council of Norway via PeTWIN (grant no. 294600) and SIRIUS (grant no. 237898).

References

1. Baramashetru, C.P., Tapia Tarifa, S.L., Owe, O., Gruschka, N.: A policy language to capture compliance of data protection requirements. In: ter Beek, M.H., Monahan, R. (eds.) Proc. 17th International Conference on Integrated Formal Methods (IFM 2022). Lecture Notes in Computer Science, vol. 13274, pp. 289–309. Springer (2022), https://doi.org/10.1007/978-3-031-07727-2_16
2. Baset, S., Stoffel, K.: Object-oriented modeling with ontologies around: A survey of existing approaches. Int. J. Softw. Eng. Knowl. Eng. **28**(11-12), 1775–1794 (2018), https://doi.org/10.1142/S0218194018400284

3. Baumann, C., Beckert, B., Blasum, H., Bormer, T.: Lessons learned from microkernel verification – specification is the new bottleneck. In: Cassez, F., Huuck, R., Klein, G., Schlich, B. (eds.) Proc. 7th Conf. on Systems Software Verification (SSV 2012). EPTCS, vol. 102, pp. 18–32 (2012), https://doi.org/10.4204/EPTCS.102.4
4. Becu, N., Neef, A., Schreinemachers, P., Sangkapitux, C.: Participatory computer simulation to support collective decision-making: Potential and limits of stakeholder involvement. Land Use Policy **25**(4), 498–509 (2008), https://linkinghub.elsevier.com/retrieve/pii/S0264837707000877
5. Blockwitz, 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: Proc. 9th International Modelica Conference. vol. 76, pp. 173–184. Linköping University Electronic Press (2012), http://dx.doi.org/10.3384/ecp12076173
6. Braberman, V.A., D'Ippolito, N., Kramer, J., Sykes, D., Uchitel, S.: MORPH: a reference architecture for configuration and behaviour self-adaptation. In: Filieri, A., Maggio, M. (eds.) Proc. 1st Intl. Workshop on Control Theory for Software Engineering (CTSE@FSE 2015). pp. 9–16. ACM (2015), https://doi.org/10.1145/2804337.2804339
7. Chaudhary, H.A.A., Guevara, I., John, J., Singh, A., Ghosal, A., Pesch, D., Margaria, T.: Model-driven engineering in digital thread platforms: A practical use case and future challenges. In: Margaria, T., Steffen, B. (eds.) Proc. 11th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2022). Lecture Notes in Computer Science, vol. 13704, pp. 195–207. Springer (2022), https://doi.org/10.1007/978-3-031-19762-8_14
8. Chekol, M.W., Euzenat, J., Genevès, P., Layaïda, N.: SPARQL query containment under SHI axioms. In: Proc. 26th AAAI Conference on Artificial Intelligence (AAAI'12). pp. 10—-16. AAAI Press (2012), https://doi.org/10.1609/aaai.v26i1.8108
9. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Serugendo, G.D.M., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software engineering for self-adaptive systems: A research roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. Lecture Notes in Computer Science, vol. 5525, pp. 1–26. Springer (2009), https://doi.org/10.1007/978-3-642-02161-9_1
10. Dahl, O.J., Nygaard, K.: SIMULA - an algol-based simulation language. Commun. ACM **9**(9), 671–678 (1966), https://doi.org/10.1145/365813.365819
11. Dalibor, M., Jansen, N., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wimmer, M., Wortmann, A.: A cross-domain systematic mapping study on software engineering for digital twins. J. Syst. Softw. **193**, 111361 (2022), https://doi.org/10.1016/J.JSS.2022.111361
12. Eramo, R., Bordeleau, F., Combemale, B., van Den Brand, M., Wimmer, M., Wortmann, A.: Conceptualizing digital twins. IEEE Software **39**(2), 39–46 (2021), https://doi.org/10.1109/MS.2021.3130755
13. Fang, H.: Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem. In: Proc. International Conference

on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER 2015). pp. 820–824. IEEE (2015), https://doi.org/10.1109/CYBER.2015.7288049
14. Fjøsna, E., Waaler, A.: READI Information modelling framework (IMF). Asset Information Modelling Framework. Tech. rep., READI Project (2021), https://readi-jip.org/wp-content/uploads/2021/03/Information-modelling-framework-V1.pdf
15. Garcia, L.F., Abel, M., Perrin, M., dos Santos Alvarenga, R.: The GeoCore ontology: A core ontology for general use in geology. Computers & Geosciences **135**, 104387 (2020), https://doi.org/10.1016/j.cageo.2019.104387
16. Gil, S., Kamburjan, E., Talasila, P., Larsen, P.G.: An architecture for coupled digital twins with semantic lifting (2024), submitted for publication
17. Gold, E.M.: Language identification in the limit. Information and Control **10**(5), 447–474 (1967), https://doi.org/10.1016/S0019-9958(67)91165-5
18. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A survey. ACM Comput. Surv. **51**(3), 49:1–49:33 (2018), https://doi.org/10.1145/3179993
19. Gruber, T.: Collective knowledge systems: Where the social web meets the semantic web. J. Web Semant. **6**(1), 4–13 (2008), https://doi.org/10.1016/j.websem.2007.11.011
20. Hai, R., Koutras, C., Quix, C., Jarke, M.: Data lakes: A survey of functions and systems. IEEE Transactions on Knowledge and Data Engineering **35**(12), 12571–12590 (Dec 2023), http://dx.doi.org/10.1109/TKDE.2023.3270101
21. Hansen, S.T., Kamburjan, E., Kazemi, Z.: Monitoring reconfigurable simulation scenarios in co-simulated digital twins. In: Proc. 12th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation. Practice (ISoLA 2024). Lecture Notes in Computer Science, Springer (2024), in production
22. Harth, A., Käfer, T., Rula, A., Calbimonte, J.P., Kamburjan, E., Giese, M.: Towards Representing Processes and Reasoning with Process Descriptions on the Web. Transactions on Graph Data and Knowledge **2**(1), 1:1–1:32 (2024), https://doi.org/10.4230/TGDK.2.1.1
23. Hitzler, P.: A review of the semantic web field. Commun. ACM **64**(2), 76–83 (2021), https://doi.org/10.1145/3397512
24. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman and Hall/CRC Press (2010), http://www.semantic-web-book.org/
25. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., Ngomo, A.N., Polleres, A., Rashid, S.M., Rula, A., Schmelzeisen, L., Sequeda, J.F., Staab, S., Zimmermann, A.: Knowledge graphs. ACM Comput. Surv. **54**(4), 71:1–71:37 (2022), https://doi.org/10.1145/3447772
26. ISO: Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM). Standard, Intl. Organization for Standardization, Geneva, CH (Mar 2018), https://www.iso.org/standard/68078.html, ISO 19650-1:2018
27. Jahandideh, I., Ghassemi, F., Sirjani, M.: An actor-based framework for asynchronous event-based cyber-physical systems. Softw. Syst. Model. **20**(3), 641–665 (2021), https://doi.org/10.1007/s10270-021-00877-y
28. Kamburjan, E.: From post-conditions to post-region invariants: deductive verification of hybrid objects. In: Bogomolov, S., Jungers, R.M. (eds.) Proc. 24th ACM Intl. Conf. on Hybrid Systems: Computation and Control (HSCC'21). pp. 9:1–9:11. ACM (2021), https://doi.org/10.1145/3447928.3456633

29. Kamburjan, E., Din, C.C.: Runtime enforcement using knowledge bases. In: Lambers, L., Uchitel, S. (eds.) Proc. 26th Intl. Conf. on Fundamental Approaches to Software Engineering (FASE 2023). Lecture Notes in Computer Science, vol. 13991, pp. 220–240. Springer (2023), https://doi.org/10.1007/978-3-031-30826-0_12
30. Kamburjan, E., Din, C.C., Schlatte, R., Tapia Tarifa, S.L., Johnsen, E.B.: Twinning-by-construction: Ensuring correctness for self-adaptive digital twins. In: Margaria, T., Steffen, B. (eds.) Proc. 11th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles ISoLA 2022). Lecture Notes in Computer Science, vol. 13701, pp. 188–204. Springer (2022), https://doi.org/10.1007/978-3-031-19849-6_12
31. Kamburjan, E., Gurov, D.: A Hoare logic for domain specification (full version). CoRR **abs/2402.00452** (2024), https://doi.org/10.48550/arXiv.2402.00452
32. Kamburjan, E., Johnsen, E.B.: Knowledge structures over simulation units. In: Martin, C.R., Emami, N., Blas, M.J., Rezaee, R. (eds.) Proc. Annual Modeling and Simulation Conf. (ANNSIM 2022). pp. 78–89. IEEE (2022), https://doi.org/10.23919/ANNSIM55834.2022.9859490
33. Kamburjan, E., Klungre, V.N., Giese, M.: Never mind the semantic gap: Modular, lazy and safe loading of RDF data. In: Groth, P., Vidal, M., Suchanek, F.M., Szekely, P.A., Kapanipathi, P., Pesquita, C., Skaf-Molli, H., Tamper, M. (eds.) Proc. 19th Intl. Conf. on the Semantic Web (ESWC 2022). Lecture Notes in Computer Science, vol. 13261, pp. 200–216. Springer (2022), https://doi.org/10.1007/978-3-031-06981-9_12
34. Kamburjan, E., Klungre, V.N., Schlatte, R., Johnsen, E.B., Giese, M.: Programming and debugging with semantically lifted states. In: Verborgh, R., Hose, K., Paulheim, H., Champin, P., Maleshkova, M., Corcho, Ó., Ristoski, P., Alam, M. (eds.) Proc. 18th Intl. Conf. on the Semantic Web (ESWC 2021). Lecture Notes in Computer Science, vol. 12731, pp. 126–142. Springer (2021), https://doi.org/10.1007/978-3-030-77385-4_8
35. Kamburjan, E., Klungre, V.N., Schlatte, R., Tapia Tarifa, S.L., Cameron, D., Johnsen, E.B.: Digital twin reconfiguration using asset models. In: Margaria, T., Steffen, B. (eds.) Proc. 11th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation. Practice (ISoLA 2022). Lecture Notes in Computer Science, vol. 13704, pp. 71–88. Springer (2022), https://doi.org/10.1007/978-3-031-19762-8_6
36. Kamburjan, E., Klungre, V.N., Tapia Tarifa, S.L., Schlatte, R., Giese, M., Cameron, D., Johnsen, E.B.: Emerging challenges in compositionality and correctness for digital twins. In: FMDT@FM. CEUR Workshop Proceedings, vol. 3507. CEUR-WS.org (2023), https://ceur-ws.org/Vol-3507/paper2.pdf
37. Kamburjan, E., Kostylev, E.V.: Type checking semantically lifted programs via query containment under entailment regimes. In: Homola, M., Ryzhikov, V., Schmidt, R.A. (eds.) Proc. 34th Intl. Workshop on Description Logics (DL 2021). CEUR Workshop Proceedings, vol. 2954. CEUR-WS.org (2021), https://ceur-ws.org/Vol-2954/paper-19.pdf
38. Kamburjan, E., Mitsch, S., Hähnle, R.: A hybrid programming language for formal modeling and verification of hybrid systems. Leibniz Trans. Embed. Syst. **8**(2), 04:1–04:34 (2022), https://doi.org/10.4230/LITES.8.2.4
39. Kamburjan, E., Schlatte, R., Johnsen, E.B., Tarifa, S.L.T.: Designing distributed control with hybrid active objects. In: Margaria, T., Steffen, B. (eds.) Proc. 9th Intl. Symp. on Leveraging Applications of Formal Methods, Verification and Validation

(ISoLA 2020). Lecture Notes in Computer Science, vol. 12479, pp. 88–108. Springer (2020), https://doi.org/10.1007/978-3-030-83723-5_7
40. Kamburjan, E., Sieve, R., Baramashetru, C.P., Amato, M., Barmina, G., Occhipinti, E., Johnsen, E.B.: GreenhouseDT: An exemplar for digital twins. In: Proc. 19th Intl. Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'24). p. 175–181. ACM (2024), https://doi.org/10.1145/3643915.3644108
41. Karabulut, E., Pileggi, S.F., Groth, P., Degeler, V.: Ontologies in digital twins: A systematic literature review. Future Gener. Comput. Syst. **153**, 442–456 (2024), https://doi.org/10.1016/j.future.2023.12.013
42. Karami, F., Basin, D.A., Johnsen, E.B.: DPL: A language for GDPR enforcement. In: Proc. 35th IEEE Computer Security Foundations Symposium (CSF 2022). pp. 112–129. IEEE (2022), https://doi.org/10.1109/CSF54842.2022.9919687
43. Kiczales, G., Rivieres, J.D.: The Art of the Metaobject Protocol. MIT Press, Cambridge, MA, USA (1991)
44. Kritzinger, W., Karner, M., Traar, G., Henjes, J., Sihn, W.: Digital twin in manufacturing: A categorical literature review and classification. IFAC-PapersOnLine **51**(11), 1016–1022 (2018), https://doi.org/10.1016/j.ifacol.2018.08.474, 16th IFAC Symp. on Information Control Problems in Manufacturing (INCOM 2018)
45. Kuruppuarachchi, P., Rea, S., McGibney, A.: Trust and security analyzer for digital twins. In: Chbeir, R., Benslimane, D., Zervakis, M.E., Manolopoulos, Y., Nguyen, N.T., Tekli, J. (eds.) Proc. 15th International Conference on Management of Digital EcoSystems (MEDES 2023). Communications in Computer and Information Science, vol. 2022, pp. 278–290. Springer (2023), https://doi.org/10.1007/978-3-031-51643-6_20
46. Lehner, D., Pfeiffer, J., Tinsel, E., Strljic, M.M., Sint, S., Vierhauser, M., Wortmann, A., Wimmer, M.: Digital twin platforms: Requirements, capabilities, and future prospects. IEEE Softw. **39**(2), 53–61 (2022), https://doi.org/10.1109/MS.2021.3133795
47. Liskov, B., Wing, J.M.: A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. **16**(6), 1811–1841 (1994), https://doi.org/10.1145/197320.197383
48. Madsen, O.L., Møller-Pedersen, B.: What object-oriented programming was supposed to be: Two grumpy old guys' take on object-oriented programming. In: Scholliers, C., Singer, J. (eds.) Proceedings of the 2022 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2022). pp. 220–239. ACM (2022), https://doi.org/10.1145/3563835.3568735
49. Madsen, O.L., Møller-Pedersen, B.: What your mother forgot to tell you about modeling - and programming. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2023 Companion. pp. 200–210. IEEE (2023), https://doi.org/10.1109/MODELS-C59198.2023.00049
50. Margaria, T., Schieweck, A.: The digital thread in industry 4.0. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) Proc. 15th Intl. Conf. on Integrated Formal Methods (iFM 2019). Lecture Notes in Computer Science, vol. 11918, pp. 3–24. Springer (2019), https://doi.org/10.1007/978-3-030-34968-4_1
51. Margaria, T., Schieweck, A.: Active behavior mining for digital twins extraction. IT Prof. **24**(4), 74–80 (2022), https://doi.org/10.1109/MITP.2022.3193044
52. Milosevic, Z., van Schalkwyk, P.: Towards responsible digital twins. In: Enterprise Design, Operations, and Computing. Lecture Notes in Business Information Processing, vol. 498, pp. 123–138. Springer (2023), https://doi.org/10.1007/978-3-031-54712-6_8

53. Pferscher, A., Wunderling, B., Aichernig, B.K., Muskardin, E.: Mining digital twins of a VPN server. In: Hallerstede, S., Kamburjan, E. (eds.) Proc. Workshop on Applications of Formal Methods and Digital Twins. CEUR Workshop Proceedings, vol. 3507. CEUR-WS.org (2023), https://ceur-ws.org/Vol-3507/paper6.pdf
54. Pichler, R., Skritek, S.: Containment and equivalence of well-designed SPARQL. In: Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 39—-50. PODS '14, Association for Computing Machinery (2014), https://doi.org/10.1145/2594538.2594542
55. Qu, Y., Kamburjan, E., Torabi, A., Giese, M.: Semantically triggered qualitative simulation of a geological process. Applied Computing and Geosciences **21**, 100152 (2024), https://doi.org/10.1016/j.acags.2023.100152
56. Qu, Y., Perrin, M., Torabi, A., Abel, M., Giese, M.: GeoFault: A well-founded fault ontology for interoperability in geological modeling. Computers & Geosciences **182**, 105478 (2024), https://doi.org/10.1016/j.cageo.2023.105478
57. Singh, S., Shehab, E., Higgins, N., Fowler, K., Reynolds, D., Erkoyuncu, J.A., Gadd, P.: Data management for developing digital twin ontology model. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture **235**(14), 2323–2337 (2021), https://doi.org/10.1177/0954405420978117
58. Smith, B.C.: Procedural Reflection in Programming Languages. Ph.D. thesis, MIT (1982), http://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TR-272.pdf
59. Talasila, P., Gomes, C., Mikkelsen, P.H., Arboleda, S.G., Kamburjan, E., Larsen, P.G.: Digital twin as a service (DTaaS): A platform for digital twin developers and users. In: 2023 IEEE Smart World Congress (SWC). pp. 1–8. IEEE (2023), https://doi.org/10.1109/SWC57546.2023.10448890
60. Wallner, F.: Development of a Robust Active Automata Learning Algorithm for Automotive Measurement Devices Avoiding Resets. Master's thesis, Graz University of Technology, Graz, Austria (2022), https://repository.tugraz.at/publications/9bn45-0d225
61. Zheng, X., Lu, J., Kiritsis, D.: The emergence of cognitive digital twin: vision, challenges and opportunities. Intl. Journal of Production Research **60**(24), 7610–7632 (2022), https://doi.org/10.1080/00207543.2021.2014591