Semantical Reflection for Computational Structures

Eduard Kamburjan
and collaborators

Antwerp, 23.11.23

University of Oslo

Reflection

What is Reflection?

- Reasoning about oneself
- Reasoning about the relation to the environment
- Forming insights: expectations and memories
- Acting on reflective insights

Reflection

What is Reflection?

- Reasoning about oneself
- Reasoning about the relation to the environment
- Forming insights: expectations and memories
- Acting on reflective insights

How can we program reflective applications?

Appearance

Beyond OO Reflection

In programming, reflection refers to the ability to manipulate runtime structures, such as classes directly – We want more:

- Reasoning about runtime structures
- Relate runtime structures to application domain
- Formulate models and data based on this relation

- How to connect a program with its application domain?
- How to interpret a program through the lens of it domain?
- How to express and adhere to domain knowledge at runtime?

Semantically Lifted Programs

Knowledge Graphs

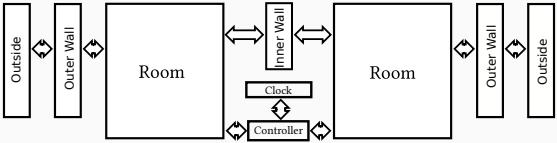
Triple-Based Knowledge Representation

Knowledge Graphs are a framework to represent (RDF), reason (OWL) over, and query (SPARQL) domain knowledge and data. Example: Asset model of a house.

Knowledge Graphs

Triple-Based Knowledge Representation

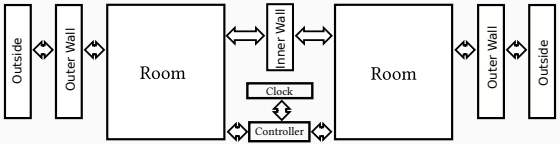
Knowledge Graphs are a framework to represent (RDF), reason (OWL) over, and query (SPARQL) domain knowledge and data. Example: Asset model of a house.



Knowledge Graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to represent (RDF), reason (OWL) over, and query (SPARQL) domain knowledge and data. Example: Asset model of a house.

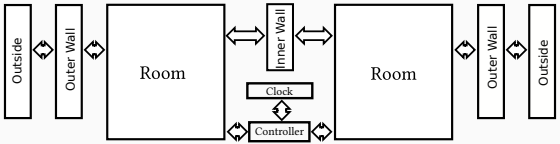



```
ast:heater1 a ast:Heater. ast:heater1 ast:in ast:room1.
ast:heater2 a ast:Heater. ast:heater2 ast:in ast:room2.
ast:heater1 ast:id 13. ast:heater2 ast:id 12.
ast:room1 ast:leftOf ast:room2.
```

Knowledge Graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to represent (RDF), reason (OWL) over, and query (SPARQL) domain knowledge and data. Example: Asset model of a house.

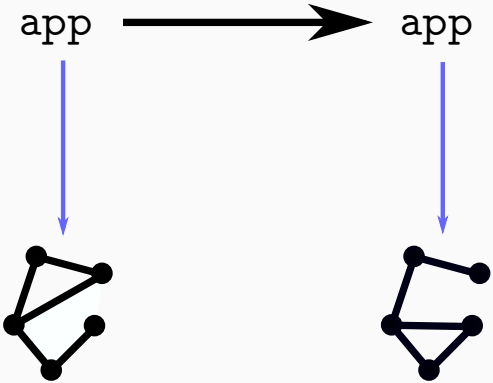



```
ast:heater1 a ast:Heater. ast:heater1 ast:in ast:room1.
ast:heater2 a ast:Heater. ast:heater2 ast:in ast:room2.
ast:heater1 ast:id 13. ast:heater2 ast:id 12.
ast:room1 ast:leftOf ast:room2.
```

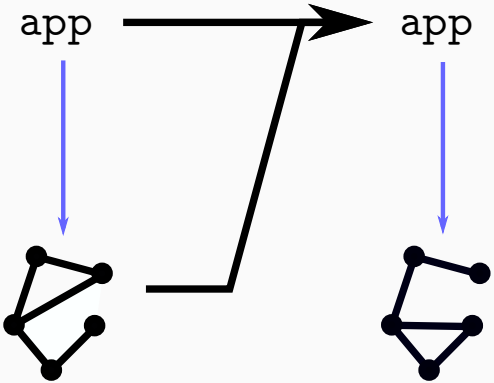
`htLeftOf` **subPropertyOf** `ast:in` **o** `ast:leftOf` **o inverse**(`ast:in`)

Semantically Lifted Programs

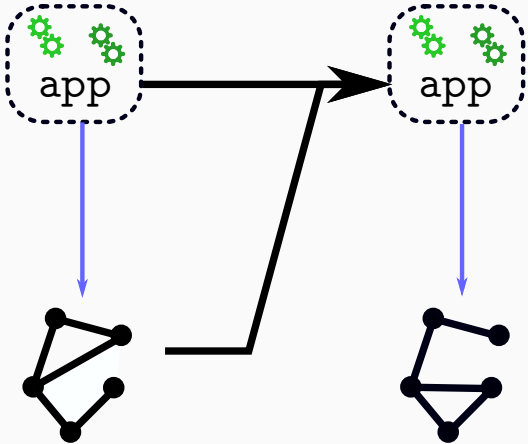
Semantically Lifted Programs



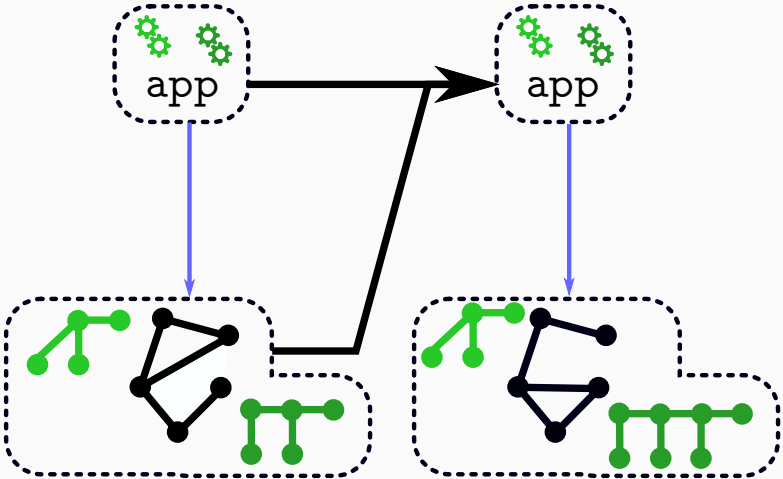
Semantically Lifted Programs



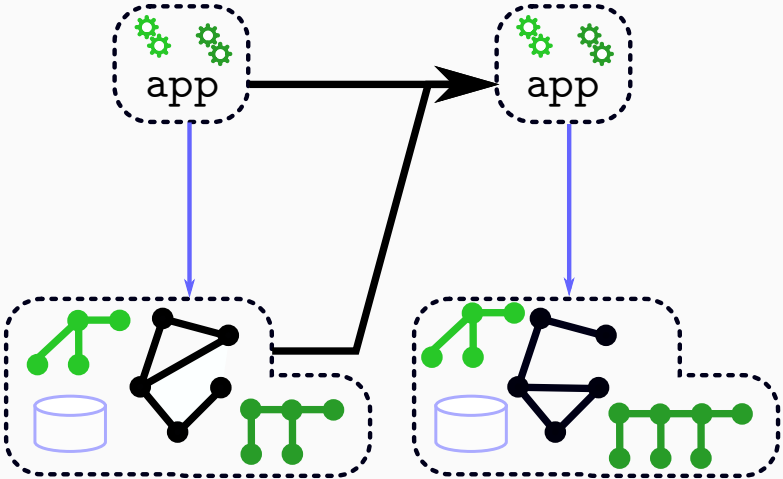
Semantically Lifted Programs



Semantically Lifted Programs



Semantically Lifted Programs



Direct Mapping of Program States

SMOL: Integration of Semantics and Semantic Technologies

Map each program state to a knowledge graph and allow program to operate on the KG. Implemented in SMOL (`smolang.org`).

```
1 class C (Int i) Unit inc(){ this.i = this.i + 1; } end
2 Main C c = new C(5);  Int i = c.inc(); end
```

Direct Mapping of Program States

SMOL: Integration of Semantics and Semantic Technologies

Map each program state to a knowledge graph and allow program to operate on the KG. Implemented in SMOL (`smolang.org`).

```
1 class C (Int i) Unit inc(){ this.i = this.i + 1; } end
2 Main C c = new C(5);  Int i = c.inc(); end
```

```
prog:C a prog:class. prog:C prog:hasField prog:i.
run:obj1 a prog:C.   run:obj1 prog:i 5.
run:proc1 a prog:process.
run:proc1 prog:runsOn run:obj1.
....
```

[K. et al., *Programming and Debugging with Semantically Lifted States*, ESWC'21]

Semantic Reflection: Reasoning about oneself

```
1 class Building(List<Room> rooms) ... end
2 class Inspector(List<Building> buildings)
3  Unit inspectStreet(String street)
4   List<Building> l := access("SELECT ?x WHERE {?x a Villa. ?x :in %
  street}");
5   this.inspectAll(l);
6  end
7 end
```

Semantic Reflection: Reasoning about oneself

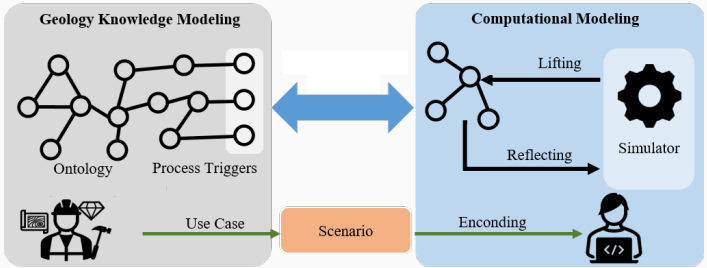
```
1 class Building(List<Room> rooms) ... end
2 class Inspector(List<Building> buildings)
3  Unit inspectStreet(String street)
4   List<Building> l := access("SELECT ?x WHERE {?x a Villa. ?x :in %
  street}");
5   this.inspectAll(l);
6  end
7 end
```

```
Villa EquivalentTo: rooms o length some xsd:int [>= 3]
```

Semantic Reflection: Reasoning about oneself – GeoSimulator

Case study of using SMOL for a geological simulator

- SMOL simulators describes the effects of the process
- SMOL state is interpreted through ontology
- Geological ontology describes under which conditions a geological process starts



[Qu, K., and Giese, *A Geological Case Study on Semantically Triggered Processes*, ESWC'23]

Semantic Reflection: Reasoning about oneself – GeoSimulator

Modeling of a geological shale structure in SMOL

```
class ShaleUnit extends GeoUnit
 (Double temperature,
  Boolean hasKerogenSource,
  Int maturedUnits)
  models
  "a GeoReservoirOntology_sedimentary_geological_object;
     location_of [a domain:amount_of_organic_matter];
     GeoCoreOntology_constituted_by [a domain:shale];
     has_quality [domain:datavalue %temperature; a domain:temperature
  ].";
end
```

Semantic Reflection: Reasoning about oneself – GeoSimulator

Resulting (part of the) knowledge graph

```
run:obj1 smol:models domain:obj1.
domain:obj1 a GeoReservoirOntology_sedimentary_geological_object;
   location_of [a domain:amount_of_organic_matter];
   GeoCoreOntology_constituted_by [a domain:shale];
   has_quality [domain:datavalue "10.0"^^xsd:Double; a domain:temperature].
```

Semantic Reflection: Reasoning about oneself – GeoSimulator

Simulation driver

```
1 List<ShaleUnit> fs =
2 member(domain:models some (obo:participates_in some domain:
  oil_window_maturation_trigger));
3 while fs != null do
4   fs.content.mature(); fs = fs.next;
5 end
```

For Mandal-Ekofisk field, simulation gives similar results as original study (2*mya* steps)

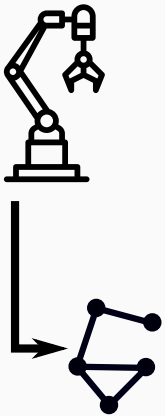
	SMOL	Cornford'94	Time Difference
Start M.	52*ma*	∼50*ma*	∼2*mya*
End M.	14*ma*	∼23*ma*	∼9*mya*
Crit. Moment	28*ma*	∼30*ma*	∼2*mya*

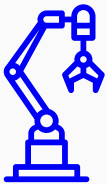
Semantic Reflection: Structurally Self-Adaptive Digital Twins

Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG

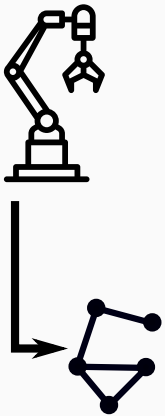


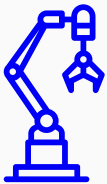


Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG

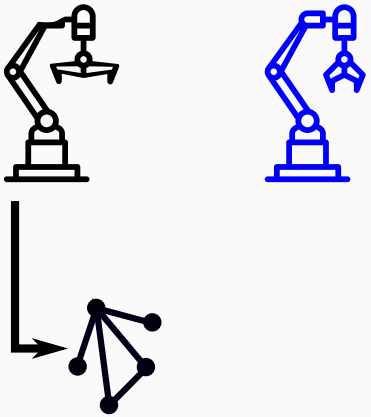




Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

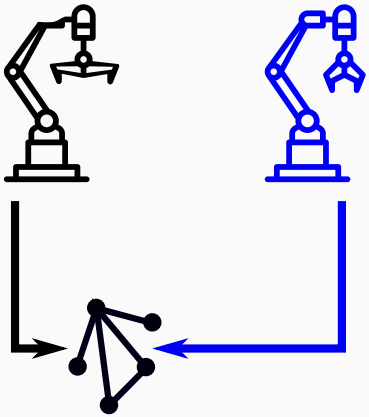
- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG



Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

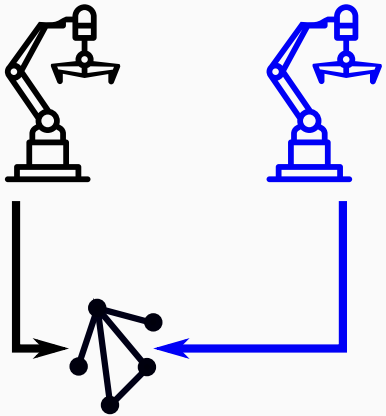
- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG



Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

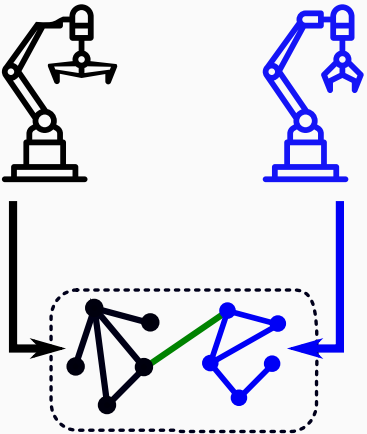
- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG



Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

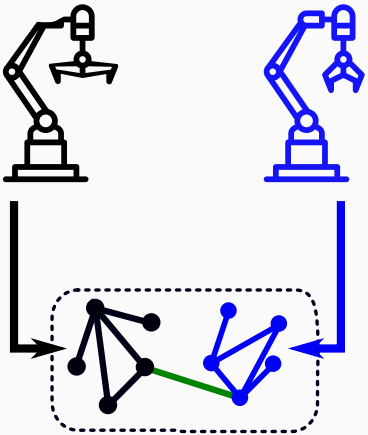
- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG



Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

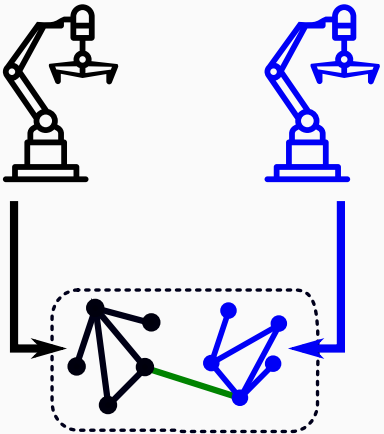
- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG



Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG



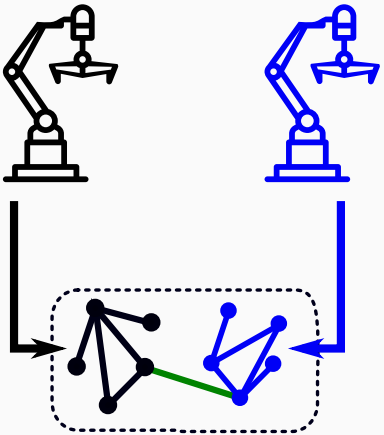
Semantic Reflection: Comparing with Expectations

Is our digital twin twinning the right thing?

- Export asset model of physical system as KG
- Export program state with simulators as KG
- Formulate constraints over combined KG

Possible Constraints

- Constraint on program
  "Is this a sensible simulation structure?"
- Constraints on twinning
  "Does the program have the same structure as the asset?"



Functional Mock-Up Interface (FMI)

Standard for (co-)simulation units, called function mock-up units (FMUs). Can also serve as interface to sensors and actuators.

Semantic Reflection: Structurally Self-Adaptive Digital Twins – SMOL/FMI

Functional Mock-Up Interface (FMI)

Standard for (co-)simulation units, called function mock-up units (FMUs). Can also serve as interface to sensors and actuators.

```
1 //simplified shadow
2 class Monitor(FMO[out Double val] sys,
3               FMO[out Double val] shadow)
4   Unit run(Double threshold)
5     while shadow != null do
6       sys.doStep(1.0);  shadow.doStep(1.0);
7       if(sys.val - shadow.val >= threshold) then ... end
8     end ...
```

Knowledge Structures over Simulation Units, K. and Johnsen. [ANNSIM'22]

Semantic Reflection: Structurally Self-Adaptive Digital Twins

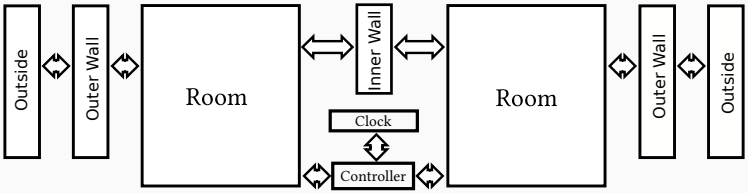
SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twins.

Semantic Reflection: Structurally Self-Adaptive Digital Twins

SPARQL

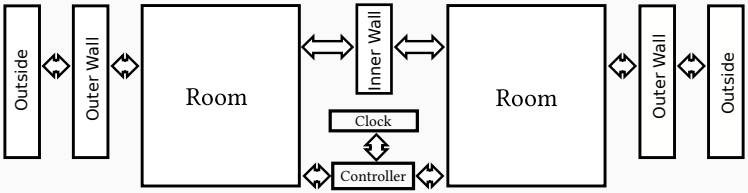
Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twins.



Semantic Reflection: Structurally Self-Adaptive Digital Twins

SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twins.




```
1 class Room(FMO f, Wall inner, Wall outer, Controller ctrl, Int id) end
2 class Controller(FMO f, Room left, Room right, Int id) end
3 class InnerWall(FMO f, Room left, Room right) end
```

Semantic Reflection: Structurally Self-Adaptive Digital Twins

SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.

Query to detect non-sensical setups:

```
SELECT ?room WHERE { ?ctrl a prog:Controller.
                     ?ctrl prog:left ?room.
                     ?ctrl prog:right ?room }
```

Semantic Reflection: Structurally Self-Adaptive Digital Twins

SPARQL

Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.

Query to check structural consistency for heaters:

```
SELECT * WHERE { ?o1 prog:id ?id1. ?h1 ast:id ?id1.
                 ?o2 prog:id ?id2. ?h2 ast:id ?id2.
                 ?h1 htLeftOf ?h2.
                 ?c a prog:Controller.
                 ?c prog:left ?o1. ?c prog:right ?o2.}
```

Semantic Reflection: Structurally Self-Adaptive Digital Twins

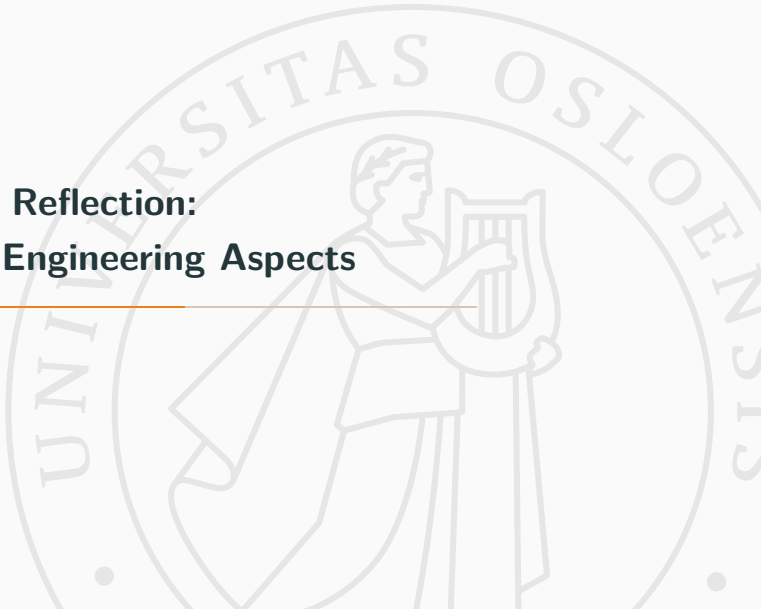
Semantic Reflection

One can use the knowledge graph *within* the program to detect structural drift: Formulate query to retrieve all mismatching parts

```
....
List<Repairs> repairs =
 construct("SELECT ?room ?wallLeft ?wallRight WHERE
   {?x ast:id ?room.
    ?x ast:right [ast:id ?wallRight].
    ?x ast:left [ast:id ?wallLeft].
    FILTER NOT EXISTS {?y a prog:Room; prog:id ?room.}}");
```

[K. et al., *Digital Twin Reconfiguration Using Asset Models*, ISoLA'22]

Semantic Reflection: Software Engineering Aspects



Software Engineering Semantic Reflection

Static Guarantees

How can we ensure that semantic reflection does not cause runtime errors?

```
1 class Building(List<Room> rooms) ... end
2 class Inspector(List<Building> buildings)
3  Unit inspectStreet(String street)
4   List<Building> l := access("SELECT ?x WHERE {?x a Villa. ?x at %
  street}");
5   this.inspectAll(l);
6  end
7 end
```

Type checking reflection reduces to query containment, if the ontology $\mathcal{K}$ is known.

$$\texttt{Villa} \sqcap \exists \texttt{at.xsd:string} \sqsubseteq_{\mathcal{K}} \texttt{Building}$$

Software Engineering Semantic Reflection

Connecting Class Models

How can we connect OWL and OO class models?

- Generate program classes from ontology
- Generate program classes for RDF structures
- Generate program classes for *queries*

Bridging the Gap

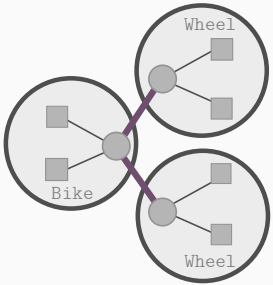
- Use retrieval queries as interface between class models
- Do not connect *concepts*, define *data retrieval*
- Annotate query to class, not execution point
- Implemented for Java, extended with Liskov Principle for subtyping

Example: Bike and Wheels

```
1 class Wheel (Int wheelId, Int year) end
2 class Bike (Int bId, Int year, Wheel front, Wheel back) end
```

Example: Bike and Wheels

```
1 class Wheel (Int wheelId, Int year) end
2 class Bike (Int bId, Int year, Wheel front, Wheel back) end
```

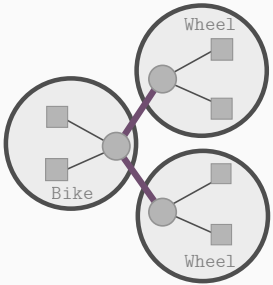



```
Q = SELECT * WHERE
  ?b :bId ?id;
     :prod ?year;
     :back ?back;
     :front ?front.
  ?back :wheelId ?wheelId1;
      :prod ?year1.
  ?front :wheelId ?wheelId2;
      :prod ?year2.
```

[K., Norstein and Giese, *Never mind the semantic gap*, ESWC'22]

Example: Bike and Wheels

```
1  List<Result> res = query(Q); Result r = res[0];
2  Wheel w1 = new Wheel(r.get("wheelId1"), r.get("year1"));
3  Wheel w2 = new Wheel(r.get("wheelId2"), r.get("year2"));
4  Bike b = new Bike(r.get("id"), r.get("year"), w1, w2);
5  print(b.front.id);
```

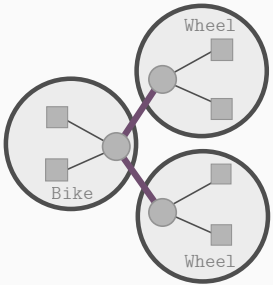



```
Q = SELECT * WHERE
  ?b :bId ?id;
     :prod ?year;
     :back ?back;
     :front ?front.
  ?back :wheelId ?wheelId1;
      :prod ?year1.
  ?front :wheelId ?wheelId2;
      :prod ?year2.
```

Example: Bike and Wheels

Challenges

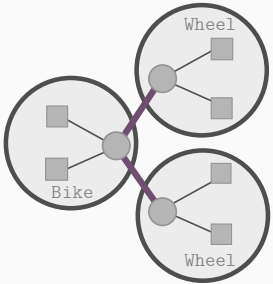
- Data access is **not type safe**
- Query is **disconnected from class**
- Query is **non-modular**: class structure is ignored, no reuse




```
Q = SELECT * WHERE
  ?b :bId ?id;
     :prod ?year;
     :back ?back;
     :front ?front.
  ?back :wheelId ?wheelId1;
      :prod ?year1.
  ?front :wheelId ?wheelId2;
      :prod ?year2.
```

Links — Detailed Explanation

```
1 class Wheel anchor ?w (Int wheelId, Int year) end
2   retrieve SELECT ?wheelId ?year { ?w :wheelId ?wheelId; :prod ?year. }
3
4 class Bike anchor ?b (Int bId;  Int year;
5   link(?b :front ?front) Wheel front;
6   link(?b :back ?back) Wheel back;
7 ) end retrieve SELECT ?id ?year { ?b :bId ?bId; :prod ?year. }
```




```
Q = SELECT * WHERE
  ?b :bId ?bId;
     :prod ?year;
     :back ?back;
     :front ?front.
  ?back :wheelId ?wheelId1;
      :prod ?year1.
  ?front :wheelId ?wheelId2;
      :prod ?year2.
```

Evaluation

Slegge

- Slegge is a query corpus for exploration in energy industry.
- Remodeling of 8 queries in extended SMOL using 27 classes.
- Found one bug due to copy-paste

Software Engineering Semantic Reflection

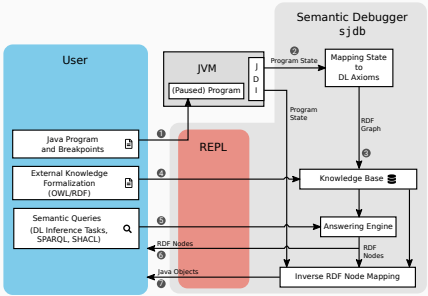
Ontologies for Programs

Two tools developed for JVM: `jdi2owl` generates a knowledge graph of a JVM state through the debugging interface. `sjdb` enables debugging of Java applications.

Software Engineering Semantic Reflection

Ontologies for Programs

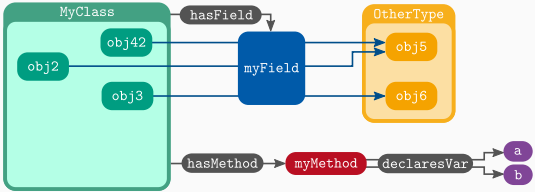
Two tools developed for JVM: `jdi2owl` generates a knowledge graph of a JVM state through the debugging interface. `sjdb` enables debugging of Java applications.



[Haubner, *Inspecting Java Program States with Semantic Web Technologies*, MSc'22]

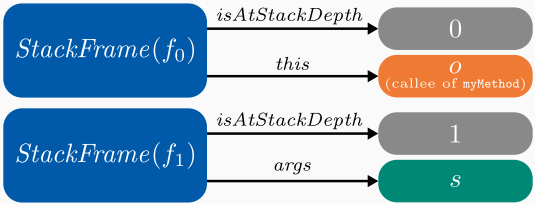
Ontologies for Programs

Two tools developed for JVM: `jdi2owl` generates a knowledge graph of a JVM state through the debugging interface. `sjdb` enables debugging of Java applications.



Ontologies for Programs

Two tools developed for JVM: `jdi2owl` generates a knowledge graph of a JVM state through the debugging interface. `sjdb` enables debugging of Java applications.



Semantically Lifted Systems

Lifting Software Architectures

Beyond Programs

- Lifting larger programs does not scale up
- Instead: Software architecture to lift only components

Lifting Software Architectures

Beyond Programs

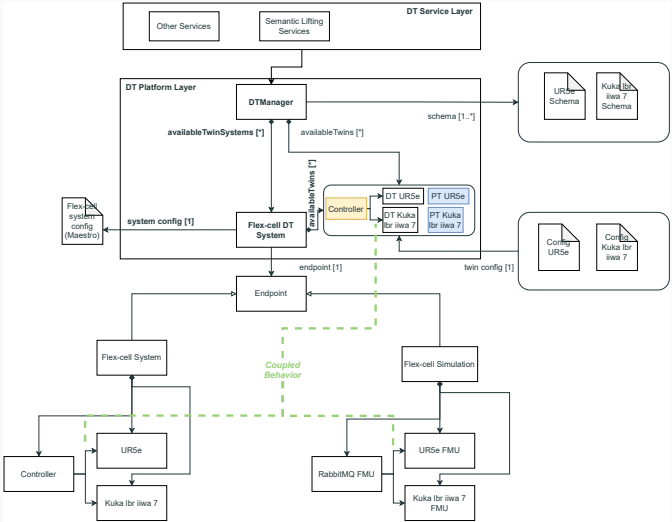
- Lifting larger programs does not scale up
- Instead: Software architecture to lift only components

[Gil, K., Talasila, Larsen, *An Architecture for Coupled Digital Twins with Semantic Lifting*, u.S.]

Lifting Software Architectures

Beyond Programs

- Lifting larger programs does not scale up
- Instead: Software architecture to lift only components



Semantic Experiment Management

Reasoning for Reuse

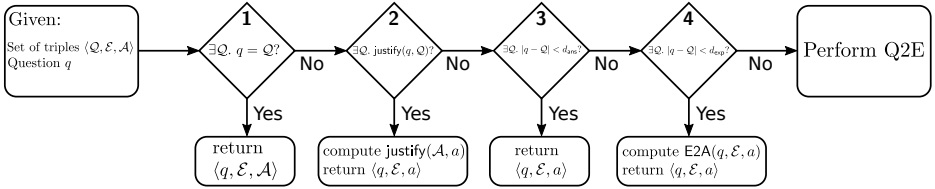
- Lifting larger programs does not scale up
- We may not be interested in the program, but computation results
- Lifting is used to detect whether reuse of computations is possible

Semantic Experiment Management

Reasoning for Reuse

- Lifting larger programs does not scale up
- We may not be interested in the program, but computation results
- Lifting is used to detect whether reuse of computations is possible

Combining Case-Based Reasoning and Deduction



[Cederbladh et al., *Symbolic Reasoning for Early Decision-Making in Model-Based Systems Engineering*, MBSE@Models'23]

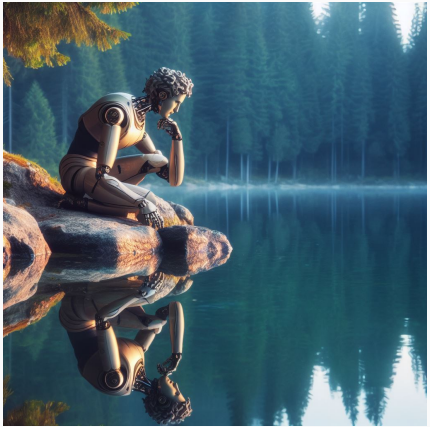
Conclusion

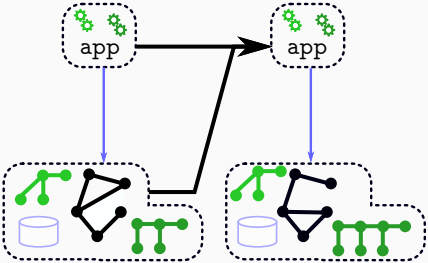
Digital Twin Lab

Digital Twin Lab

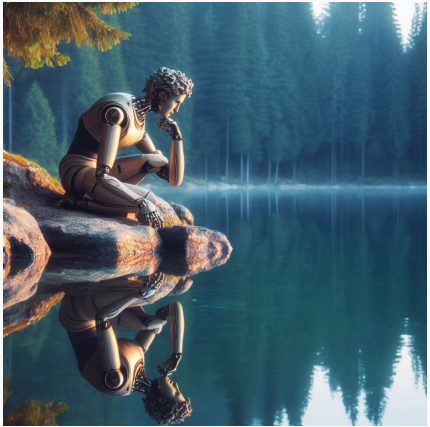
- Working with realistic software stack
- Evaluation of proposed architectures
- Software engineering with reflective and heterogenous programs

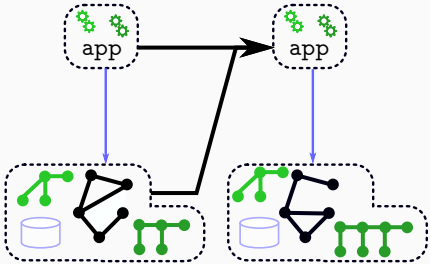
Conclusion





Conclusion





Thank you for your attention