Digital Twin Reconfiguration Using Asset Models

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

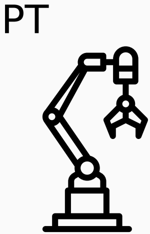
University of Oslo
26.10.2022, ISoLA 2022

Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.

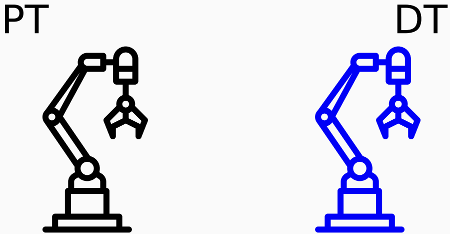
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



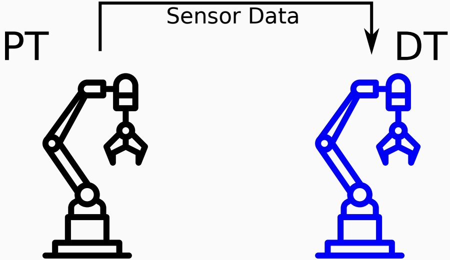
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



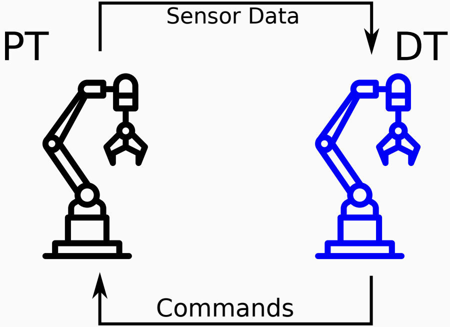
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



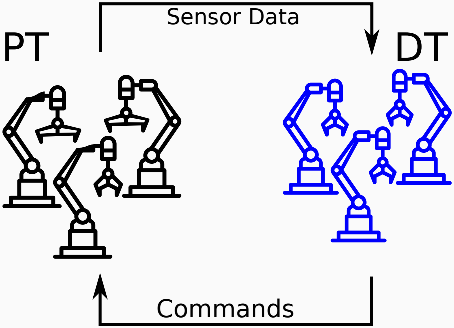
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



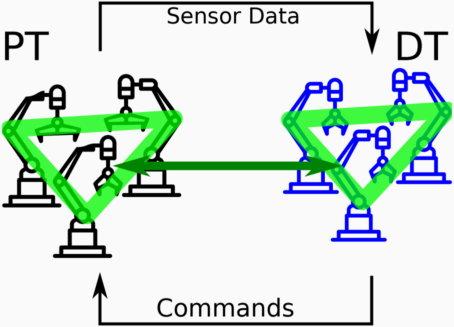
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



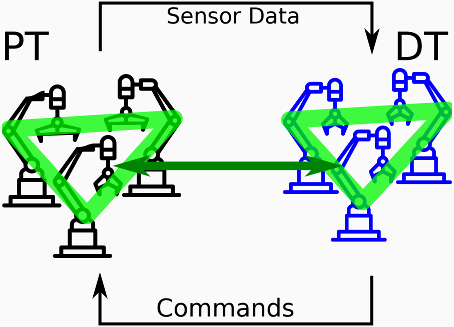
Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



Digital Twin

A digital twin system connects a physical asset with its own (simulation) models using data streams and commands.



- How to access structure of DT?
- How to express twinning?

Knowledge Graphs and Asset Models

Asset Model

An asset model is an organized, digital description of the composition and properties of a physical asset.

For example, an inventory enriched with spatial information, design plans, . . . Several projects specific to digital twins, e.g., the Asset Administration Shell of the Industry 4.0.

Our Asset Model

A knowledge graph describing the structure of the physical twin.

Using Semantic Technologies for Uniform Data Access and integration of domain knowledge.

Checking the Twinning Property

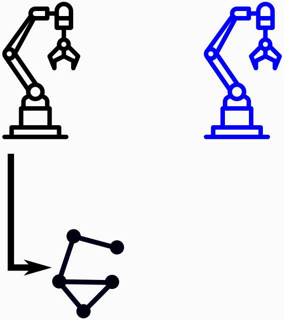
Combining the Knowledge

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

Checking the Twinning Property

Combining the Knowledge

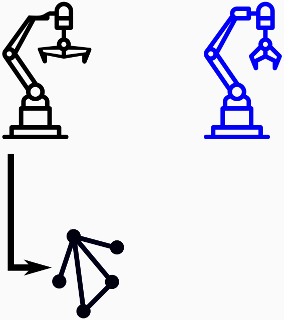
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

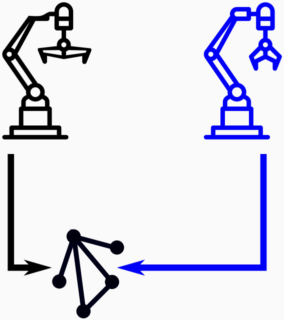
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

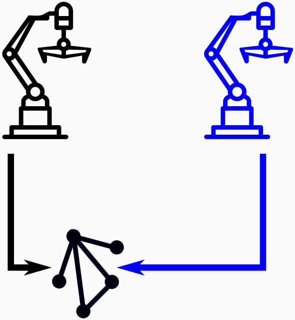
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

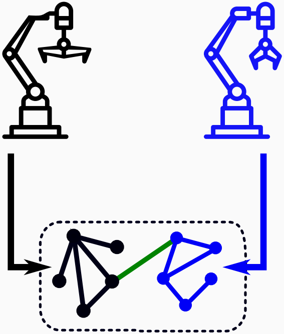
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

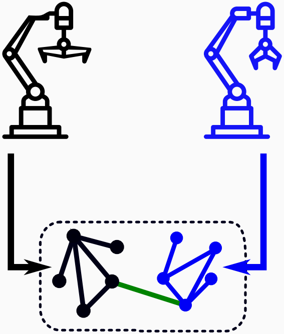
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

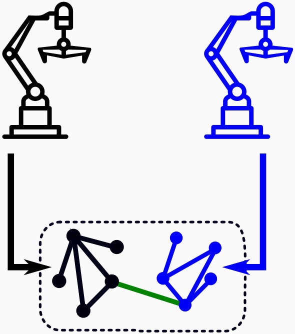
- Export asset model of physical system as knowledge graph
- Export program state with simulators as knowledge graph
- Formulate constraints over combined knowledge



Checking the Twinning Property

Combining the Knowledge

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



Checking the Twinning Property

Combining the Knowledge

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

Possible Constraints

- Constraint on asset model

  "Is the asset model consistent?"
- Constraint on program

  "Is this a sensible simulation structure?"
- Constraints on twinning

  "Does the program have the same structure as the asset?"

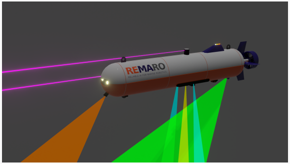
Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems, e.g., for robotics



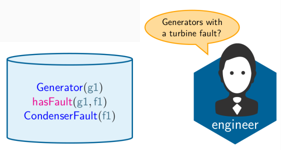
REMARO
RELIABLE AI FOR MARINE ROBOTICS

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry

Optique



Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry
- Reasoning for expert systems
  e.g., in the biomedical field

GENEONTOLOGY
Unifying Biology

SNOMED CT
The global
language of
healthcare

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry
- Reasoning for expert systems
  e.g., in the biomedical field
- Data integration
  e.g., as industrial standards

IEEE SA
STANDARDS
ASSOCIATION

READI

Knowledge Representation with Ontologies

Ontologies are logically formalized domain knowledge

- Intelligence for autonomous systems,
  e.g., for robotics
- Data access for domain experts
  e.g., in the energy industry
- Reasoning for expert systems
  e.g., in the biomedical field
- Data integration
  e.g., as industrial standards

IEEE SA
STANDARDS ASSOCIATION

READI

Surrounding theories and tools are Semantic Technologies

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Peter a Person. Paul a Person. Maria a Person.
Peter hasChild Paul. Paul hasChild Maria.
```

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF: `Peter a Person. Paul a Person. Maria a Person.`
`Peter hasChild Paul. Paul hasChild Maria.`

OWL: `hasChild` **some** (`hasChild` **some** `Person`)
subClassOf `GrandParent`

Semantic Technologies

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.

RDF:

```
Peter a Person. Paul a Person. Maria a Person.
Peter hasChild Paul. Paul hasChild Maria.
```

OWL: `hasChild` **some** (`hasChild` **some** `Person`)
subClassOf `GrandParent`

SPARQL: `SELECT ?x WHERE { ?x a GrandParent }`

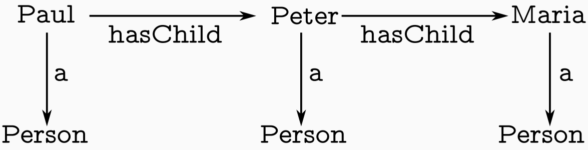
Knowledge Graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.



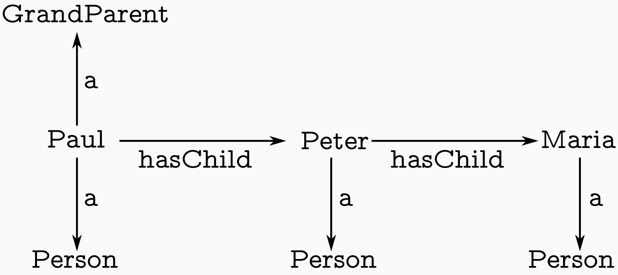
Knowledge Graphs

Triple-Based Knowledge Representation

Knowledge Graphs are a framework to (a) represent, (b) reason over, and (c) query domain knowledge and data.

W3C Standards

RDF for data, OWL for knowledge, SPARQL for queries.



GrandParent
a
Paul
hasChild
Peter
hasChild
Maria
a
a
a
Person
Person
Person

Semantically Lifted Programs and Digital Twins

Semantically Lifted States

A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.

Semantically Lifted States

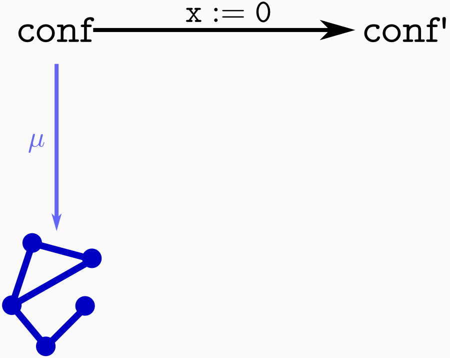
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.

$$\text{conf} \xrightarrow{\texttt{x} := 0} \text{conf'}$$

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

Semantically Lifted States

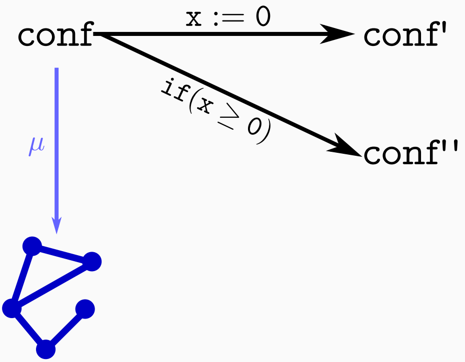
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

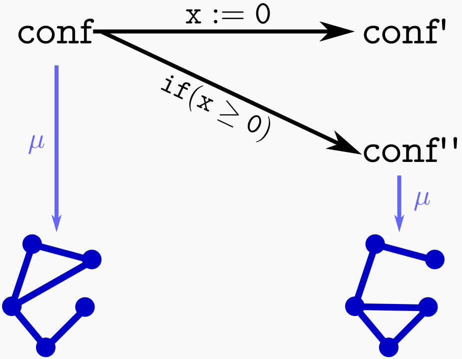
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

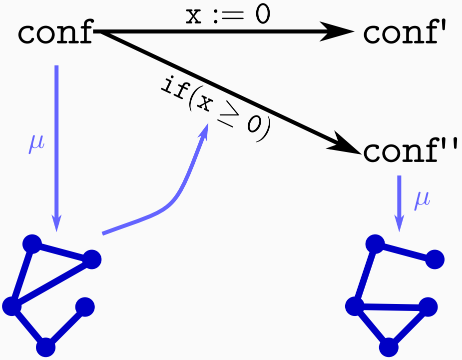
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

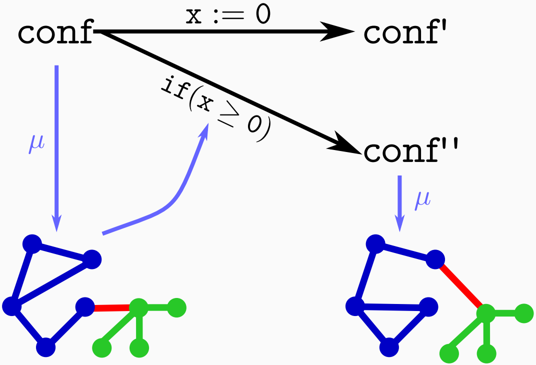
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Example

```
class C (Int i) Unit inc() this.i = this.i + 1; end end
main C c = new C(5);  Int i = c.inc(); end
```

Example

```
class C (Int i) Unit inc() this.i = this.i + 1; end end
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.
....
```

Example

```
class C (Int i) Unit inc() this.i = this.i + 1; end end
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.
....
```

A representation of (a) the full AST and (b) the full runtime state.

SMOL

Semantic Micro Object Language

Implementation of semantical lifting in an interpreted language. Type system and REPL for debugging available. (try it at **www.smolang.org**)

Given the lifted state, we can use it for multiple operations.

- **Access it** to retrieve objects without traversing pointers.
- **Enrich it** with an ontology, perform logical reasoning and retrieve objects using a query *using the vocabulary of the domain*.
- **Combine it** with another knowledge graph and access external data based on information from the current program state.

Semantic Programming

```
class Platform(List<Server> serverList) ... end
class Server(List<Task> taskList) ... end
class Scheduler(List<Platform> platformList)
 Unit reschedule()
  List<Platform> l
    := access("SELECT ?x WHERE {?x a :Overloaded}");
  this.adaptPlatforms(l);
 end
end
```

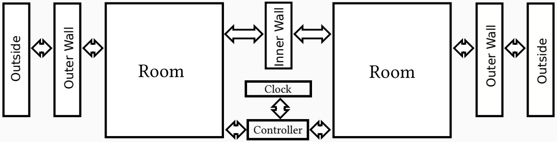
Semantic Programming

```
class Platform(List<Server> serverList) ... end
class Server(List<Task> taskList) ... end
class Scheduler(List<Platform> platformList)
 Unit reschedule()
  List<Platform> l
    := access("SELECT ?x WHERE {?x a :Overloaded}");
  this.adaptPlatforms(l);
 end
end
```

```
:Overloaded
 owl:equivalentClass [
   owl:onProperty (:tasks, :length);
   owl:minValue 3;
 ].
```

Example

Back to digital twins

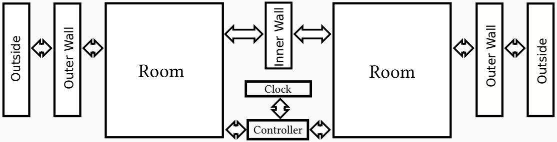


Our Asset Model

A knowledge graph describing the structure of the physical twin.

Example

Back to digital twins



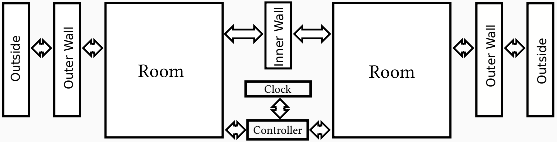
Our Asset Model

A knowledge graph describing the structure of the physical twin.

```
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.
```

Example

Back to digital twins



Our Asset Model

A knowledge graph describing the structure of the physical twin.

```
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`)

SMOL and 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.

SMOL and 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.

```
//simplified shadow
class Monitor(Cont[out Double val] sys,
              Cont[out Double val] shadow)
  Unit run(Double threshold)
    while shadow != null do
      sys.doStep(1.0);  shadow.doStep(1.0);
      if(sys.val - shadow.val >= threshold) then ... end
    end ...
```

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

Constraints on Digital Twins

Semantically Lifting the Digital Twin

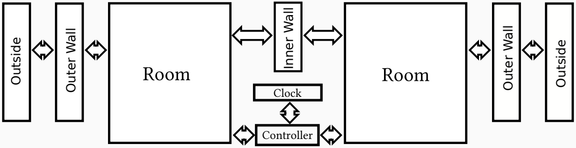
SPARQL

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

Semantically Lifting the Digital Twin

SPARQL

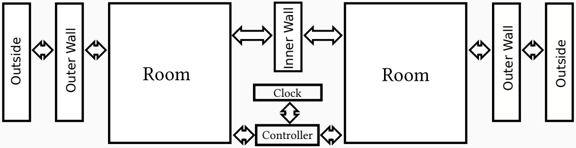
Define structural requirements as queries in SPARQL on *combined* knowledge graph, to use domain constraints on digital twin.



Semantically Lifting the Digital Twin

SPARQL

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



```
class Room(Cont[...] f,
           Wall inner, Wall outer, Controller ctrl,
           Int id) end
class Controller(Cont[...] f,
                 Room left, Room right, Int id) end
class InnerWall(Cont[...] f, Room left, Room right) end
```

Semantically Lifting the Digital Twin

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 }
```

Semantically Lifting the Digital Twin

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.}
```

Repairing your Twin

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.}}");
```

Repair function must restore structure.

Demo

`Repair`

Conclusion

Conclusion

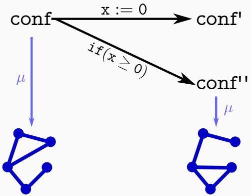
Digital Twins and Asset Models

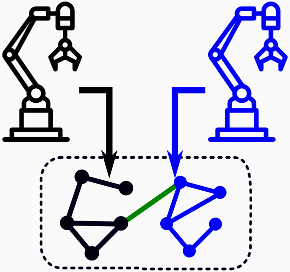
- Knowledge graph for uniformity
- Combining knowledge representation and programming
- Fully formal setting for digital twins
- Today 17:00 XbyC track: Digital Thread and Monitoring

Conclusion

Digital Twins and Asset Models

- Knowledge graph for uniformity
- Combining knowledge representation and programming
- Fully formal setting for digital twins
- Today 17:00 XbyC track: Digital Thread and Monitoring





Conclusion

Digital Twins and Asset Models

- Knowledge graph for uniformity
- Combining knowledge representation and programming
- Fully formal setting for digital twins
- Today 17:00 XbyC track: Digital Thread and Monitoring