The Semantically Reflected Digital Twin

ICTAC Summer School Tutorial 2022

Einar Broch Johnsen
Silvia Lizeth Tapia Tarifa
Rudolf Schlatte
Eduard Kamburjan

University of Oslo

Yesterday:

- **Part I** Digital Twins Introduction:
  Concepts and Engineering Perspectives
- **Part II** Modelling Knowledge using Semantic Technologies

Today:

- **Part III Modelling Physical Systems**
- **Part IV** Semantically Reflected Digital Twins

Physical Model

Summary: Yesterday

- Concept of digital twins
- Logic and ontologies for asset models

Next Part

Integration of open, industrial standards for simulation and interfaces.

- How to integrate simulations?
  Modelica for physical modeling
- How to interface with sensors and actuators?
  Functional Mock-Up Interface (FMI)

Later

Putting it all together

Outline

- Modeling physical systems: a brief introduction to Modelica
- The FMI standard: generating black-box simulators ("FMUs") from Modelica models
- Talking to simulators within a SMOL model

Outline (practical)

We will do the following:

- Model a water tank in Modelica;
- Compile the Modelica model into an FMU;
- Write a simple SMOL program that loads the FMU;
- Simulate and control the system's behavior over time

Software we will be using

- SMOL, a language for integrating knowledge bases and simulations
  - See `http://smolang.org`
  - Code at `https://github.com/smolang/SemanticObjects`
- Modelica: a language for modeling physical systems
  - download from `https://www.openmodelica.org/?id=78`
  - `https://modelica.readthedocs.io/`
  - `https://www.openmodelica.org`

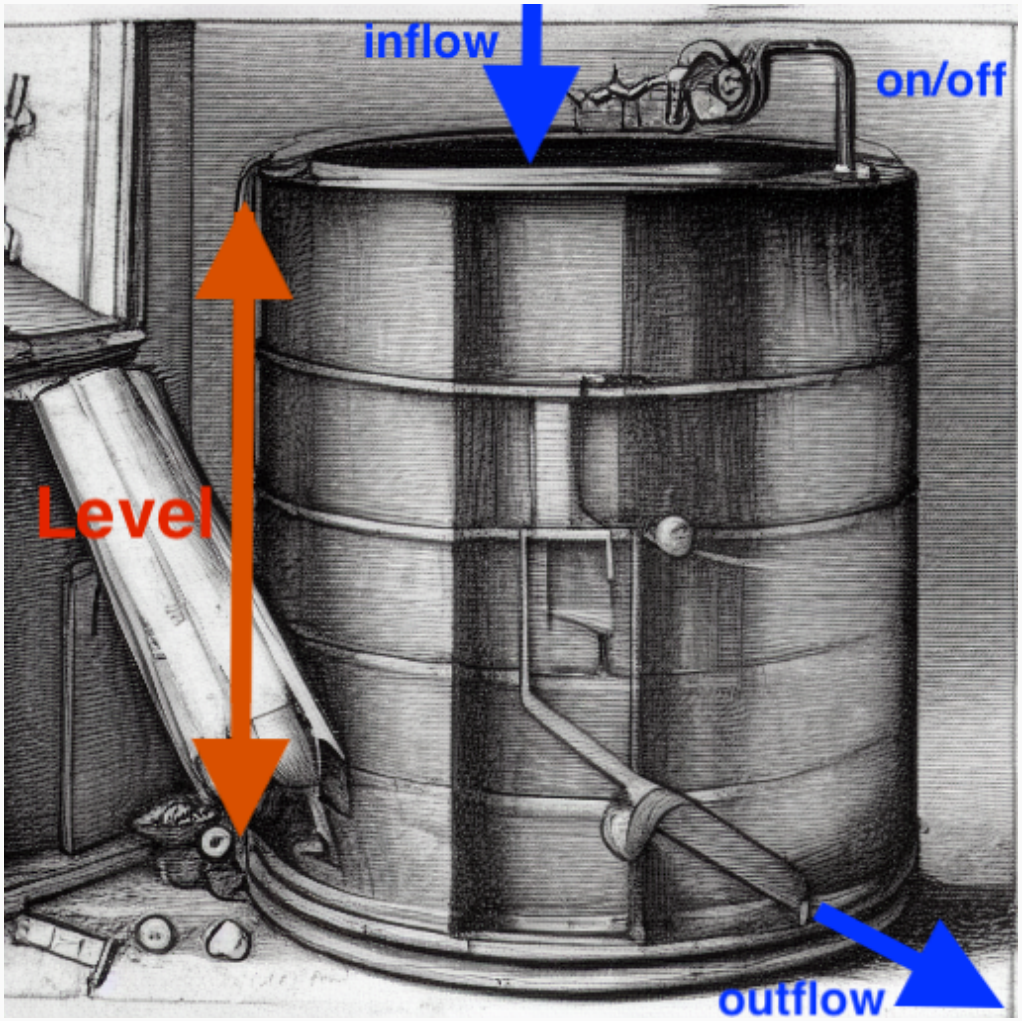
A water tank, as designed by AI

"A sketch of a watertank, a faucet on top fills the tank with water, water flows out from a hole at the bottom of the tank, by Albrecht Dürer"

A water tank, as designed by AI

"A sketch of a watertank, a faucet on top fills the tank with water, water flows out from a hole at the bottom of the tank, by Albrecht Dürer"

A water tank, as designed by AI



"A sketch of a watertank, a faucet on top fills the tank with water, water flows out from a hole at the bottom of the tank, by Albrecht Dürer"

A water tank, behaviorally

l current level in tank

d drain rate, “size of hole”

f rate of inflow

v valve control (0 or 1)

l’ the rate of change in level

A water tank, behaviorally

l current level in tank

d drain rate, "size of hole"

f rate of inflow

v valve control (0 or 1)

l' the rate of change in level

Behavior of the tank

$$l' = -d * l + v * f$$

The water tank in Modelica

```
model Tank
  parameter Real d = 0.5          "drain rate";
  parameter Real f = 2.0          "rate of inflow";
  input Boolean v(start = false) "Valve closed / open";
  output Real l(start = 5)        "water level";
  Real inFlow                     "Current fill rate";
equation
  der(l) = ( - d) * l + inFlow;
  if v then inFlow = f; else inFlow = 0.0; end if;
end Tank;
```

Modelica

- Run watertank in openmodelica (omedit)

. . . Demo . . .

Black-Box Simulators: The FMI Standard

- A digital twin combines multiple models of its subsystems, as expressed by the asset model
- `https://fmi-standard.org` ("The leading standard to exchange dynamic simulation models") shows how to combine FMUs ("functional mock-up units")
- Modelica can create FMUs from models
- SMOL can control FMUs

Generating an FMU

- Generate within OMEdit: Right-click, select "Export → FMU"
- Generate from the command line: Run `omc generate_fmu.mos`, with the file `generate_fmu.mos` containing the following:

```
installPackage(Modelica);
loadModel(Modelica);
loadFile("simple_tank.mo");
buildModelFMU(Tank, version="2.0", fmuType="me_cs");
getErrorString()
```

Running the FMU inside SMOL

- SMOL can *intantiate* FMUs, via the `simulate` expression.
- FMUs can be seen as peculiar objects, with writable and readable fields and a `tick` method to advance time.
- The names of the fields come from the metadata inside the FMU

Running the FMU inside SMOL (2)

```
main
    FMO[in Boolean v, out Double l] de
      = simulate("Tank.fmu", v = False);
    print(de.l);
    while de.l > 2 do
        de.tick(1.0);
        print(de.l);
    end
    de.v = True;
    while de.l < 5 do
        de.tick(1.0);
        print(de.l);
    end
end
```

Running the FMU inside SMOL

Demo ...

Yesterday:

- **Part I** Digital Twins Introduction:
  Concepts and Engineering Perspectives
- **Part II** Modelling Knowledge using Semantic Technologies

Today:

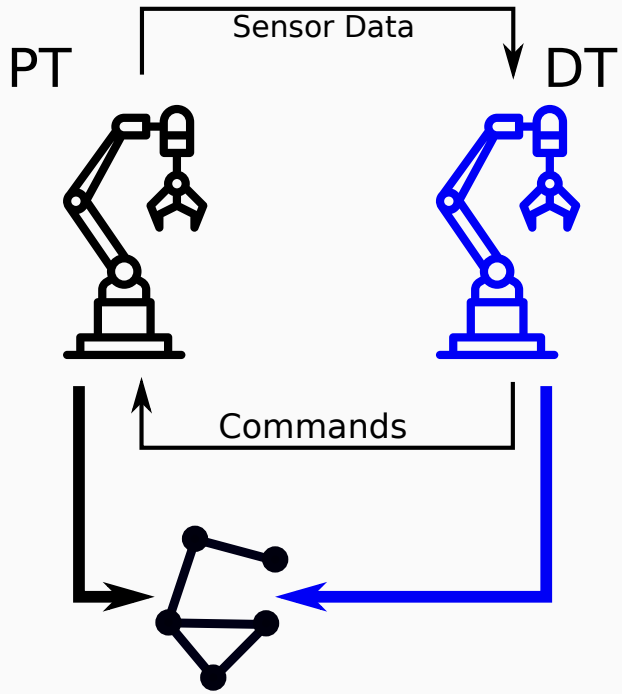
- **Part III** Modelling Physical Systems
- **Part IV Semantically Reflected Digital Twins**

Structural Self-Adaptation

- We can access the sensors of the physical system (FMI),
- access the structure of the physical system (SWT), and
- simulate the digital design (FMI).

Structural Self-Adaptation

- We can access the sensors of the physical system (FMI),
- access the structure of the physical system (SWT), and
- simulate the digital design (FMI).

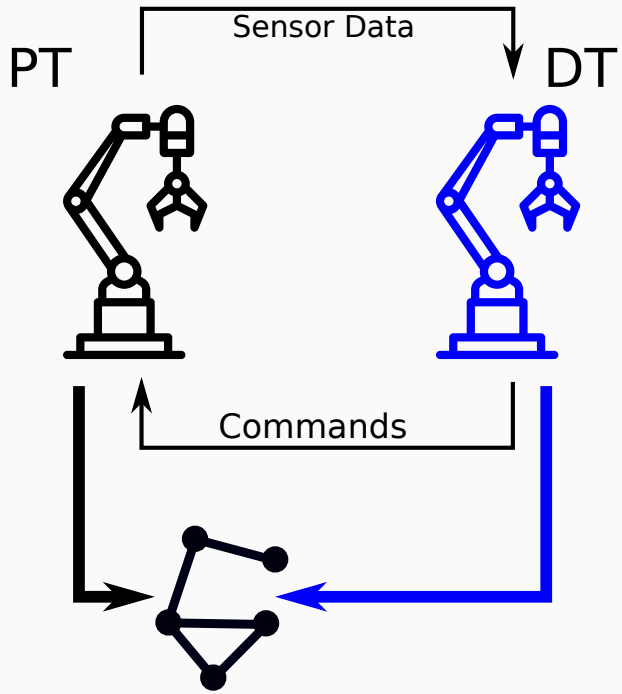


Structural Self-Adaptation

- We can access the sensors of the physical system (FMI),
- access the structure of the physical system (SWT), and
- simulate the digital design (FMI).

Putting it all together

- Compare simulations to sensors
- Compare digital with physical structure
- Self-adapt to changes in physical system

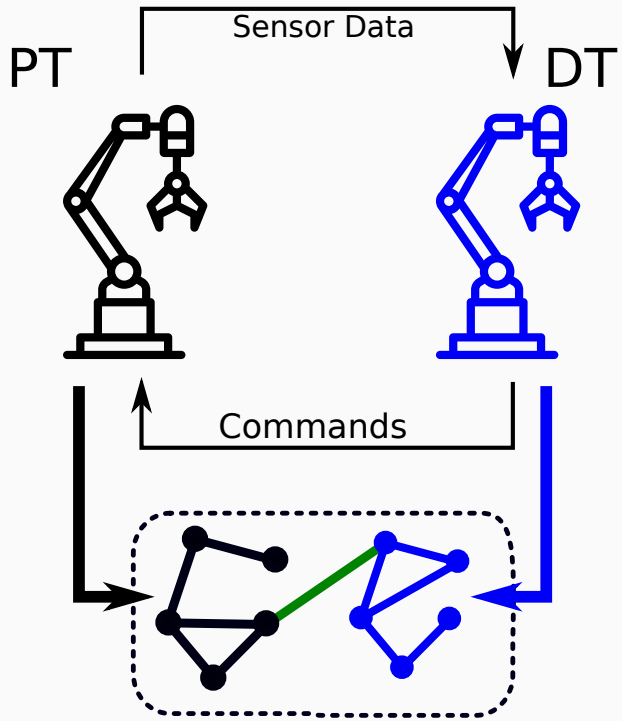


Structural Self-Adaptation

- We can access the sensors of the physical system (FMI),
- access the structure of the physical system (SWT), and
- simulate the digital design (FMI).

Putting it all together

- Compare simulations to sensors
- Compare digital with physical structure
  What is the digital structure?
- Self-adapt to changes in physical system

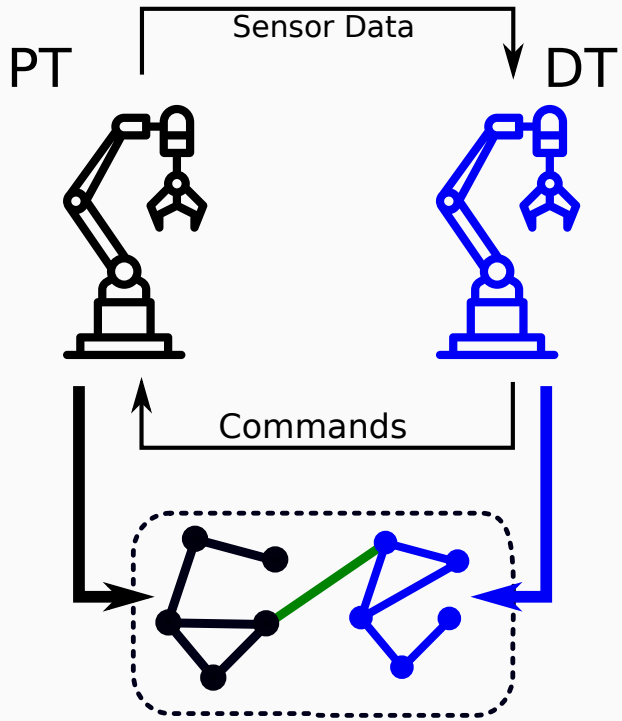


Structural Self-Adaptation

- We can access the sensors of the physical system (FMI),
- access the structure of the physical system (SWT), and
- simulate the digital design (FMI).

Putting it all together

- Compare simulations to sensors
- Compare digital with physical structure
  What is the digital structure?
- Self-adapt to changes in physical system
  Self-adaptation?



Self-Adaptation (I)

Digital Twins: Self-Adaptation

Self-adaptation means to *automatically* reestablish some property of a system, by reacting to outside stimuli. For Digital Twins, the "outside" is the physical system.

Two kinds of self-adaptation to reestablish the *twinning* property:

- Behavioral self-adaptation if sensors and simulators mismatch
- Structural self-adaptation if *structures* mismatch

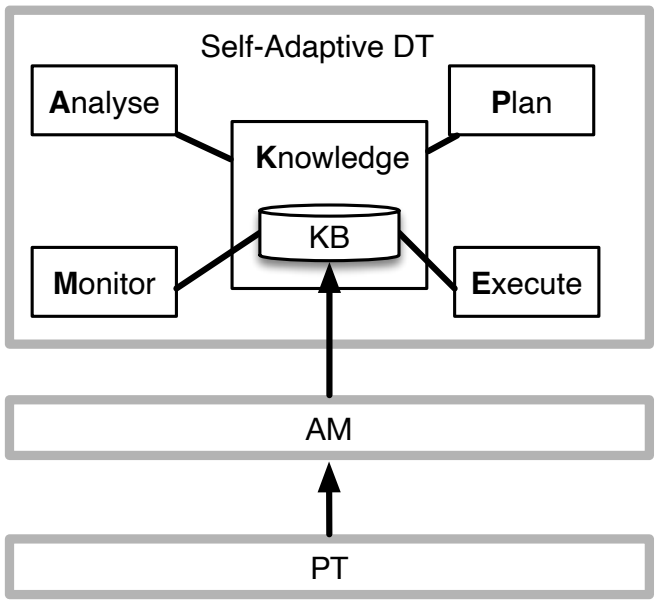
MAPE-K

MAPE-K is an established conceptual framework to structure self-adaptive systems.

MAPE-K

MAPE-K is an established conceptual framework to structure self-adaptive systems.

- A **K**nowledge component keeps track of information and goals for the self-adaptation loop:
- **M**onitor the situation
- **A**nalyze whether the situation requires adaptation
- **P**lan the adaptation
- **E**xecute the plan



Self-Adaptation (II)

Behavioral Self-Adaptation

Simulated (=expected) behavior of certain components does not match the real (=measured) behavior of the sensors.

- Monitor sensors
- Analyze the relation to simulation
- Plan repair by, e.g., finding new simulation parameters
- Exchange simulators or send signal to physical system

Reasons

- Sensor drift
- Modeling errors
- Faults
- Unexpected events

Self-Adaptation (III)

Structural Self-Adaptation

Simulated structure of digital system does not match real (= expressed in asset model) structure.

Self-Adaptation (III)

Structural Self-Adaptation

Simulated structure of digital system does not match real (= expressed in asset model) structure.

Semantically Lifted Programs

We need to express the program structure, so we can *uniformly* access it together with the asset model. How to apply semantic web technologies on programs? $\Rightarrow$ Semantical lifting.

Self-Adaptation (III)

Structural Self-Adaptation

Simulated (= lifted) structure of digital system does not match real (= expressed in asset model) structure.

Semantically Lifted Programs

We need to express the program structure, so we can *uniformly* access it together with the asset model. How to apply semantic web technologies on programs? $\Rightarrow$ Semantical lifting.

Semantical lifting is a mechanism to automatically generate the knowledge graph of a program state.

Self-Adaptation

Repair

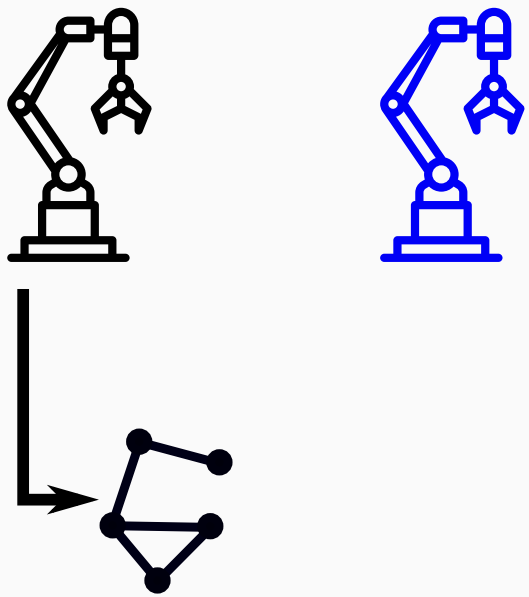
To self-adapt we must (1) detect broken twinning and (2) repair it.

Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model

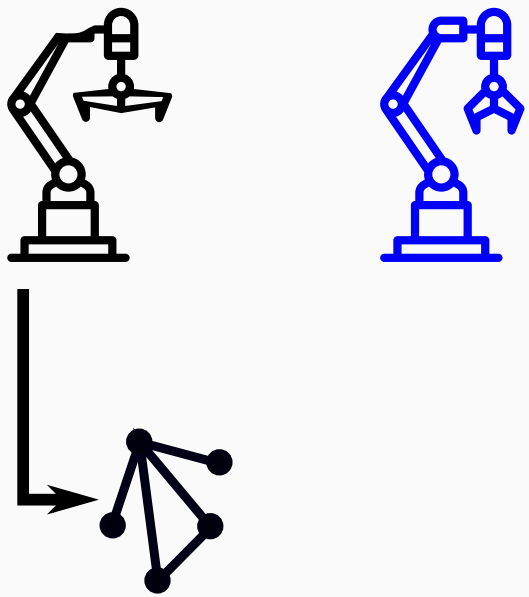


Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model
- Changes of PT are visible in asset model

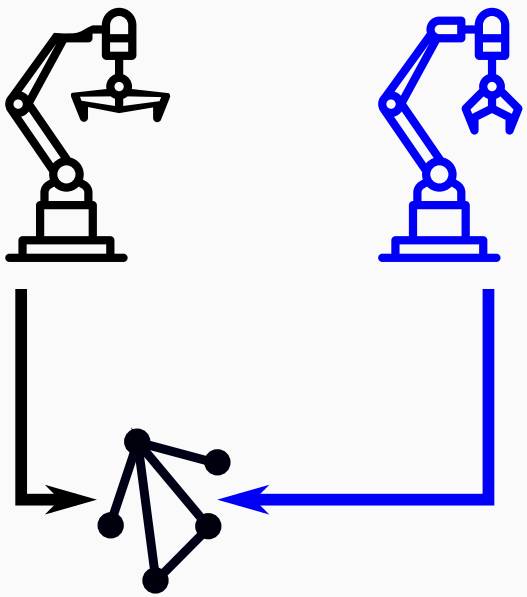


Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model
- Changes of PT are visible in asset model
- Asset model accessible directly to DT

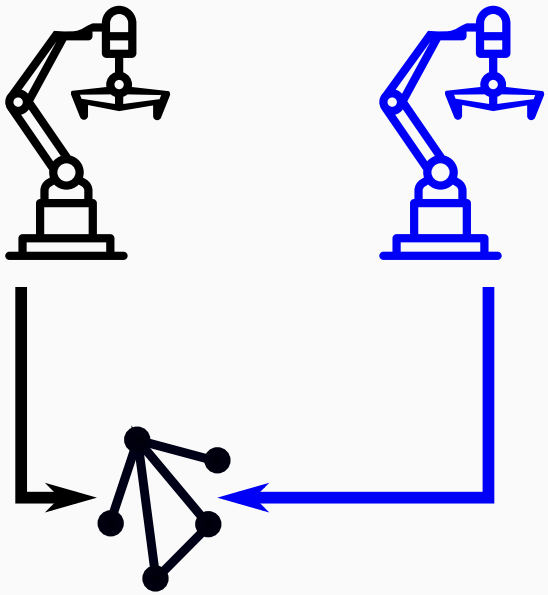


Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model
- Changes of PT are visible in asset model
- Asset model accessible directly to DT

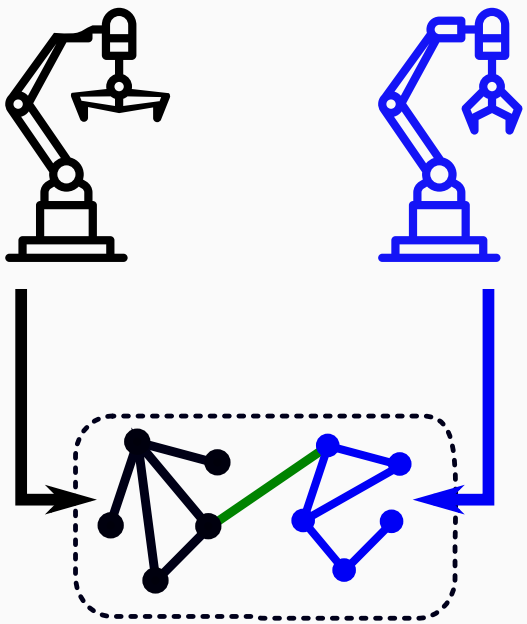


Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model
- Changes of PT are visible in asset model
- Asset model accessible directly to DT
- Detect changes through combined knowledge graph

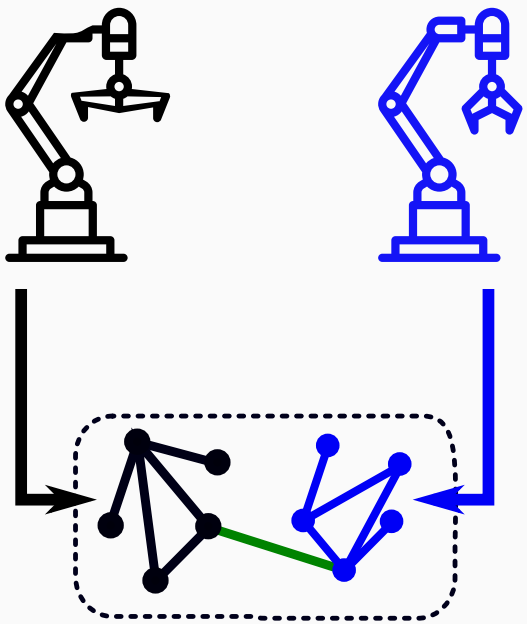


Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model
- Changes of PT are visible in asset model
- Asset model accessible directly to DT
- Detect changes through combined knowledge graph

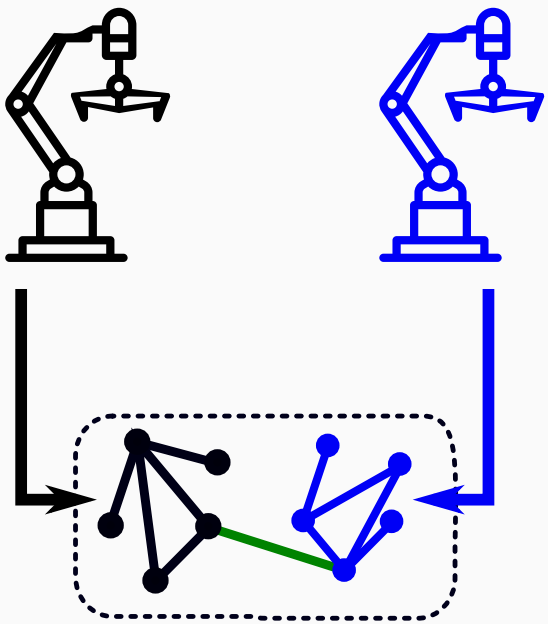


Self-Adaptation

Repair

To self-adapt we must (1) detect broken twinning and (2) repair it.

- Access PT structure through asset model
- Changes of PT are visible in asset model
- Asset model accessible directly to DT
- Detect changes through combined knowledge graph
- Information for repair available there!

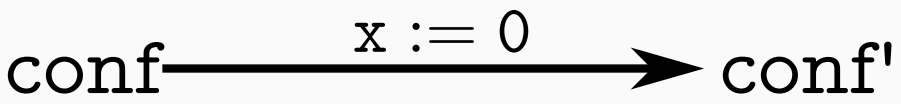


Semantically Lifted States

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

Semantically Lifted States

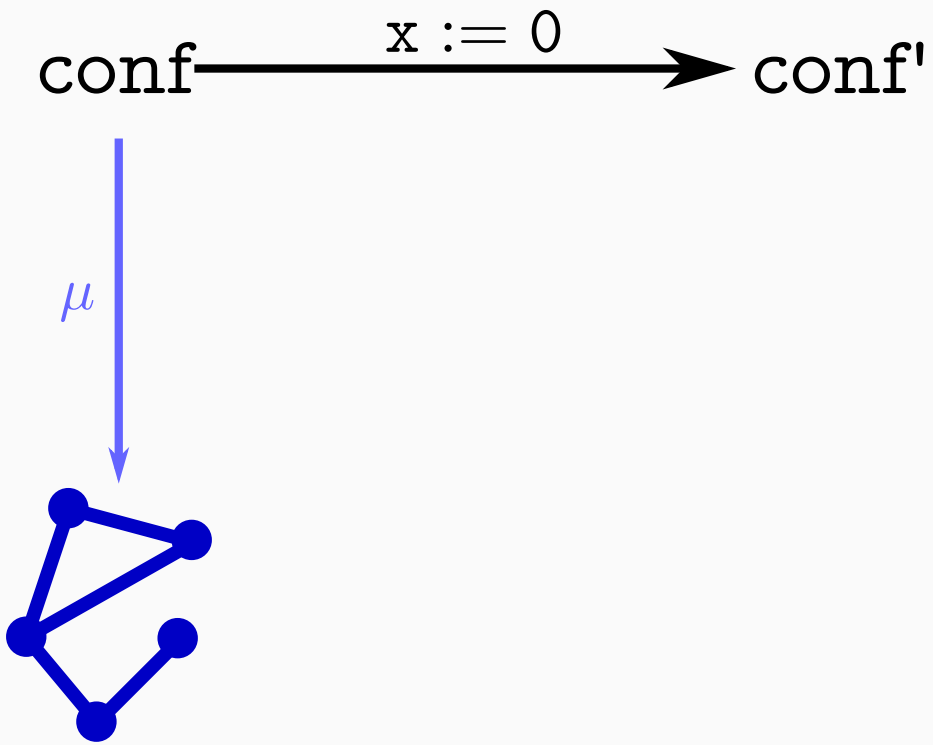
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

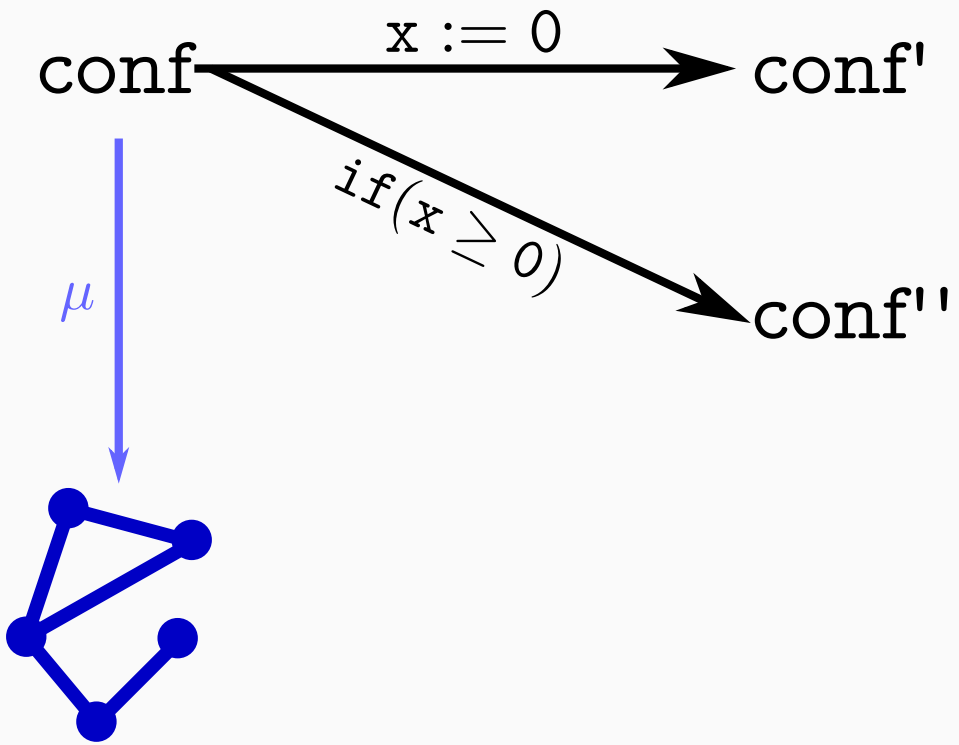
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

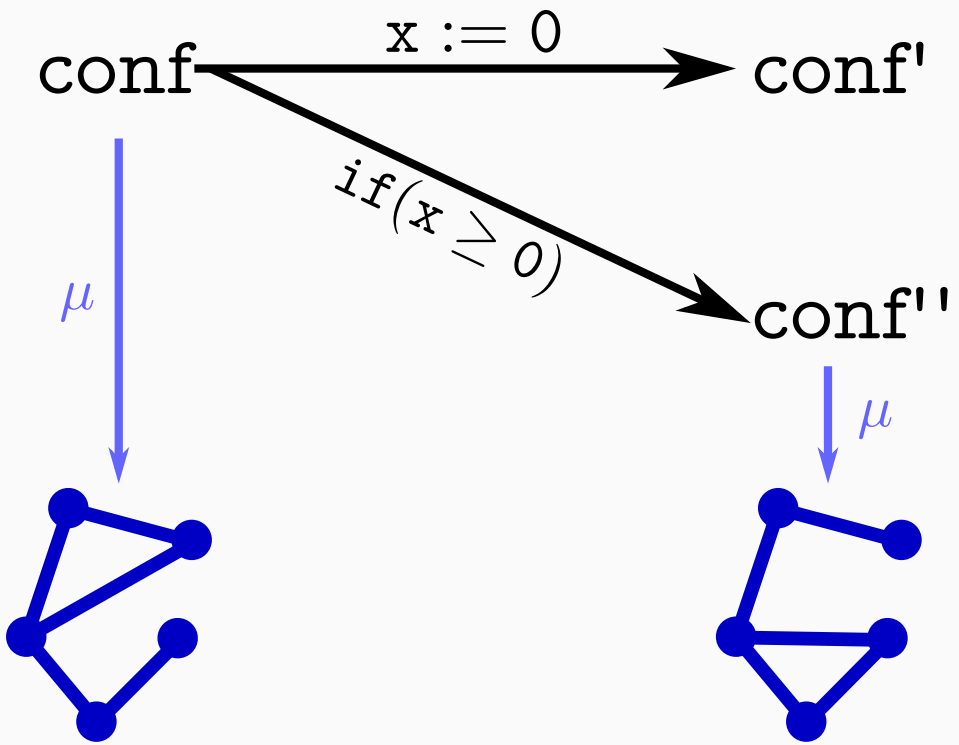
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

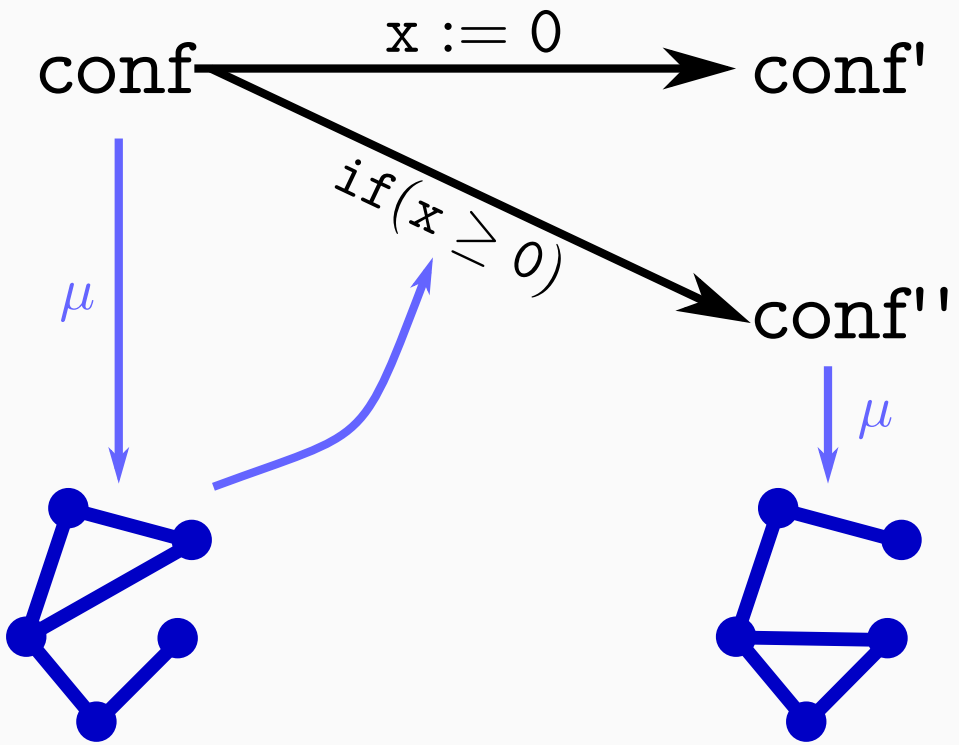
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

Semantically Lifted States

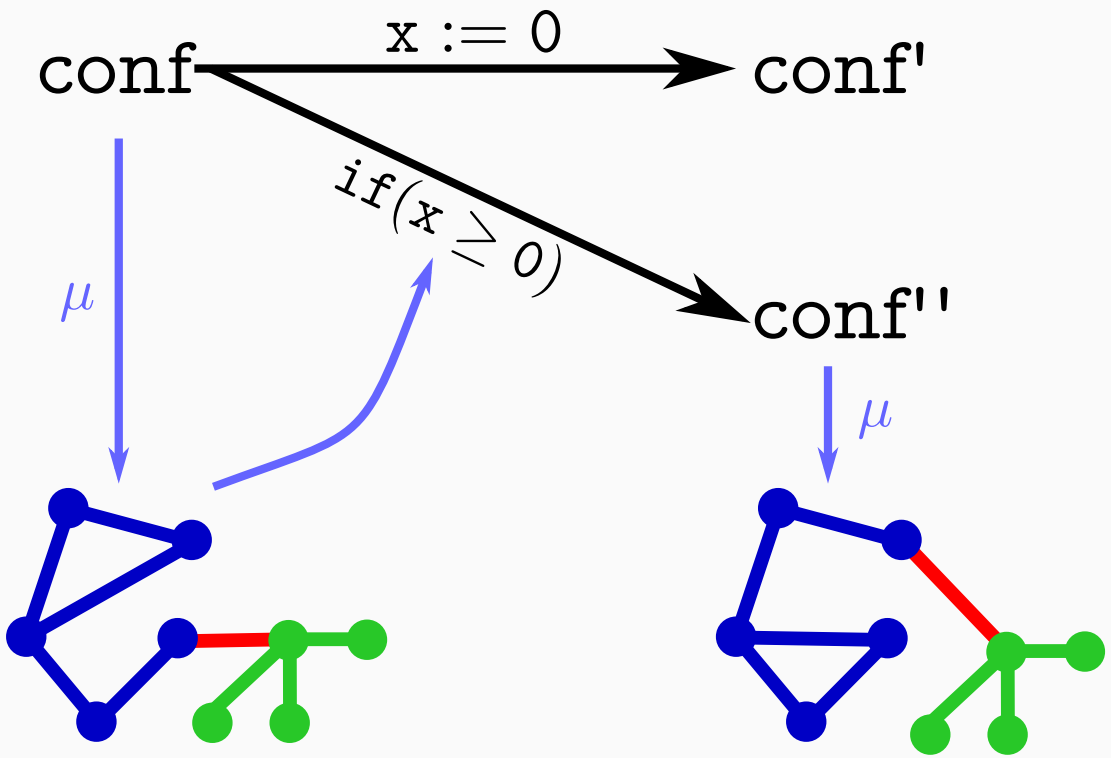
A semantically lifted program can interpret its own program state as a knowledge graph and reflect on itself through it.



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

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

Example

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

```
prog:C a prog:class. prog:C prog:hasField prog:C_i.
run:obj1 a prog:C.   run:obj1 prog:C_i 5.
...
prog:C prog:hasMethod prog:C_inc.
prog:inc prog:hasBody prog:s;
...
run:stack run:top run:frame1. run:frame1 run:executes prog:inc.
...
```

SMOL

Implementation

Semantical lifting and reflection is implemented in the **S**emantic **M**icro **O**bject **L**anguage, `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

```
1 class Server(List<Task> taskList) ... end
2 class Scheduler(List<Platform> serverList)
3   Unit reschedule()
4     List<Server> l
5       := access("SELECT ?x WHERE {?x a :Overloaded}");
6     this.adapt(l);
7   end
8 end
```

Semantic Programming

```
1 class Server(List<Task> taskList) ... end
2 class Scheduler(List<Platform> serverList)
3   Unit reschedule()
4     List<Server> l
5       := access("SELECT ?x WHERE {?x a :Overloaded}");
6     this.adapt(l);
7   end
8 end
```

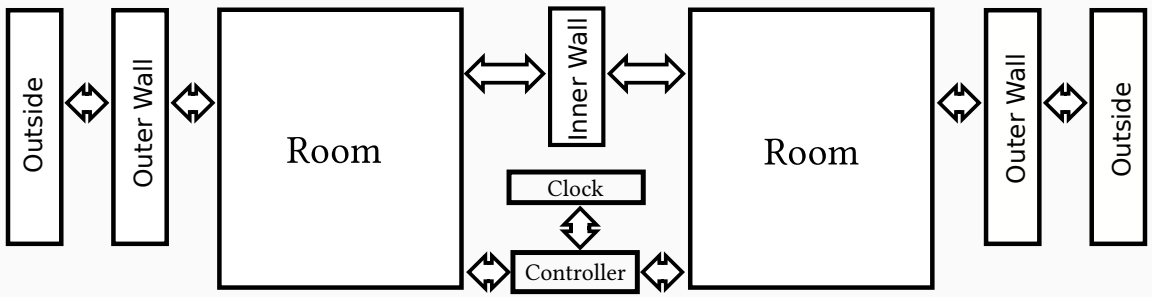
```
:Overloaded
  owl:equivalentClass [
    owl:onProperty (:taskList, :length);
  owl:minValue 3;
].
```

Demo

Semantic Reflection

Example

Back to digital twins



- Monitor consistency
- Monitor twinning
- Adapt to addition of new rooms

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

Ensuring Correctness for Self-Adaptive Digital Twins, Kamburjan et al. [ISoLA'22]

Model Description

```
<fmiModelDescription fmiVersion="2.0" modelName="Example" ...>
  <CoSimulation needsExecutionTool="true" .../>
  <ModelVariables>
    <ScalarVariable name="p" variability="continuous"
                    causality="parameter">
      <Real start="0.0"/>
    </ScalarVariable>
    <ScalarVariable name="input" variability="continuous"
                    causality="input">
      <Real start="0.0"/>
    </ScalarVariable>
    <ScalarVariable name="val" variability="continuous"
                    causality="output" initial="calculated">
      <Real/>
  </ModelVariables>
  <ModelStructure> ... </ModelStructure>
</fmiModelDescription>
```

SMOL and FMI

Functional Mock-Up Objects (FMOs)

Tight integration of simulation units using FMI into programs.

```
1 //setup
2 FMO[out Double val] shadow =
3     simulate("Sim.fmu", input=sys.val, p=1.0);
4 FMO[out Double val] sys = simulate("Realsys.fmu");
5 Monitor m = new Monitor(sys,shadow); m.run(1.0);
```

SMOL and FMI

Functional Mock-Up Objects (FMOs)

Tight integration of simulation units using FMI into programs.

```
1 //setup
2 FMO[out Double val] shadow =
3     simulate("Sim.fmu", input=sys.val, p=1.0);
4 FMO[out Double val] sys = simulate("Realsys.fmu");
5 Monitor m = new Monitor(sys,shadow); m.run(1.0);
```

Integration

- Type of FMO directly checked against model description
- Variables become fields, functions become methods
- Causality reflected in type

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

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.

```
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 ...
```

Is this twinning something? Is this setup correctly?

SMOL and FMI

SMOL with FMOs

FMOs are objects, so they are part of the knowledge graph.

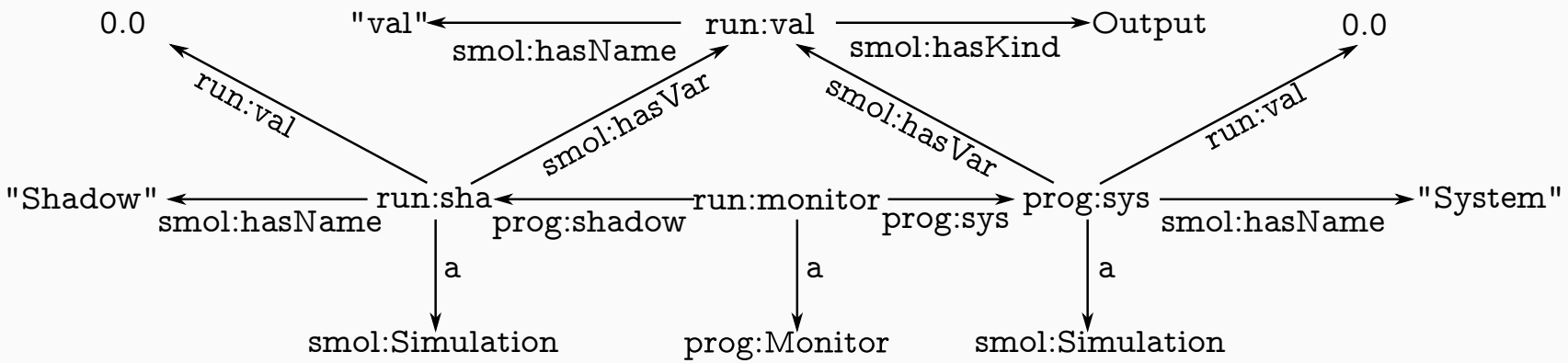
```
1 class Monitor(FMO[out Double val] sys,
2               FMO[out Double val] shadow)
```

SMOL and FMI

SMOL with FMOs

FMOs are objects, so they are part of the knowledge graph.

```
1 class Monitor(FMO[out Double val] sys,
2                FMO[out Double val] shadow)
```



Using the Semantical Lifting

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:Controller_left ?room.
                     ?ctrl prog:Controller_right ?room }
```

Using the Semantical Lifting

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:Room_id ?id1. ?h1 asset:id ?id1.
                 ?o2 prog:Room_id ?id2. ?h2 asset:id ?id2.
                 ?h1 htLeftOf ?h2.
                 ?c a prog:Controller.
                 ?c prog:Controller_left ?o1.
                 ?c prog:Controller_right ?o2}
```

Demo

Inconsistent Twinning

Self-Adapting to Structural Drift

Detecting Structural Drift

The previous query can detect that some mismatch between asset model and program state exists.
How to detect where the mismatch is and how to repair it?

- Retrieve all assets, and their connections by id (**M**)
- Remove all ids present in the digital twin
- If any id is left, assets needs to be twinned (**A**)
- Find kind of defect to plan repair (**P**)
- Execute repair according to connections (**E**)
- Monitor connections using previous query
- (And v.v. to detect twins that must be removed)

Example: Adding a New Room

- Get all (asset) rooms and their neighboring walls
- Remove all (twinned) rooms with the same id
- Use the information about walls to
- Assumption: at least one new room is next to an existing one

```
1 class RoomAsrt(String room, String wallLt, String wallRt) end
2 ....
3 List<RoomAsrt> newRooms =
4  construct(" SELECT ?room ?wallLt ?wallRt WHERE
5   { ?x a asset:Room;
6        asset:right [asset:Wall_id ?wallRt];
7        asset:left [asset:Wall_id ?wallLt]; asset:Room_id ?room.
8     FILTER NOT EXISTS {?y a prog:Room; prog:Room_id ?room.} }");
```

Demo

Repair

Remarks

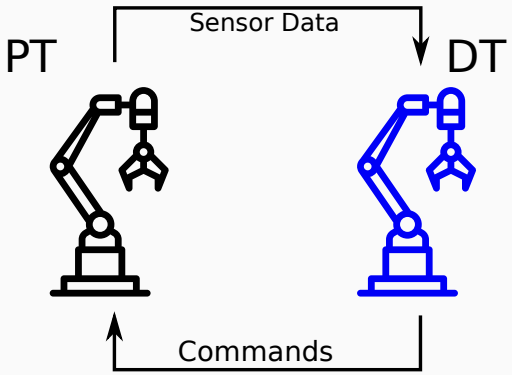
Assumptions

- We know all the possible modifications up-front
  E.g., how to deal with a heater getting new features?
- We know how to always correct structural drift
- Changes do not happen faster than we can repair

Monitoring is still needed to (a) ensure that repairs work correctly, and (b) detect loss of twinning due to, e.g., unexpected structural drift.

Wrap-Up

Summary

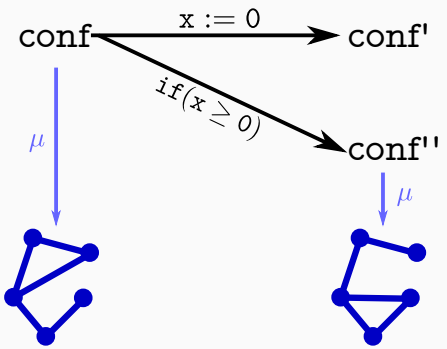


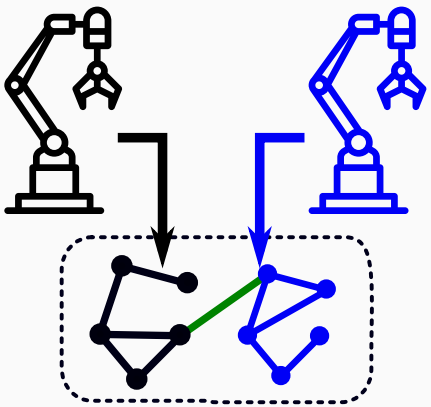
Digital Twins and the FMI

Modern systems with interconnected physical asset and digital model.

Semantic Lifting and Asset Models

Interpret program state as knowledge graph to connect with asset model – use industrial standards.





Structural Self-Adaptation

Use semantic technologies to query and monitor combined knowledge graph from asset model and program state.

Summary

What have we used to construct a self-adaptive, semantically reflected Digital Twin?

Technologies

- Semantic Web technologies
  - OWL/Protege
  - RDF, SPARQL
- Physical modeling, interfacing
  - Modelica, FMI
- SMOL

Concepts

- Digital Twins
- Self-Adaptation through MAPE-K loop
- Semantically lifted programs
- Asset models

Current Research Questions

Digital Twins and Formal Methods

- How to use the fully formal setting for static analysis?
- How to generate digital twins automatically?
- How to deal with concurrency?

Current Research Questions

Digital Twins and Formal Methods

- How to use the fully formal setting for static analysis?
- How to generate digital twins automatically?
- How to deal with concurrency?

Digital Twins@UiO

If you are interested in semantic technologies for programs or digital twins, contact us under

$$\texttt{X@ifi.uio.no},\ \texttt{X} \in \{\text{einarj, sltarifa, rudi, eduard}\}$$

Current Research Questions

Digital Twins and Formal Methods

- How to use the fully formal setting for static analysis?
- How to generate digital twins automatically?
- How to deal with concurrency?

Digital Twins@UiO

If you are interested in semantic technologies for programs or digital twins, contact us under

`X@ifi.uio.no`, $\texttt{X} \in \{\text{einarj, sltarifa, rudi, eduard}\}$

Thank you for your attention