The Self-Aware Digital Twin

Einar Broch Johnsen

University of Oslo
einarj@ifi.uio.no

BCS-FACS webinar
15 October 2024

http://www.sirius-labs.no

Talk Overview

Digital Twins: Self-Aware Model-Centric Systems

- What are digital twins and why should they be self-aware?
- How can we program self-aware digital twins?
- How do digital twins adapt to changes in the twinned systems?
- What is correctness for digital twins?

What are digital twins?

Digital Twins — The Hype

Digital twins are an emerging, enabling technology for industry to transition to the next level of digitisation

Increasing traction of digital twins

1. A means to **understand** and **control** assets in nature, industry, and society at large
2. Companies increasingly create digital twins of their physical assets

Success stories

1. GE experienced 5–7% increase of energy production from digitizing wind farms
2. Johns Hopkins Hospital's centre for clinical logistics reported 80% reduction of operating theatre holds due to delays
3. For the Johan Sverdrup oil field, digital twin innovations have boosted earnings by $216 million in one year

Digital Twins: A Best Practice Engineering Discipline

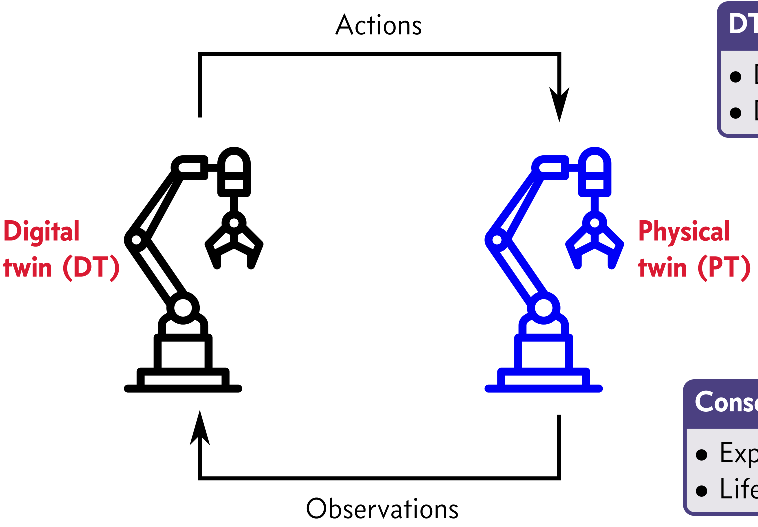
- DTs originally conceived at NASA for the space program.
- DTs have emerged as an engineering discipline, based on **best practices**

NASEM's definition of a DT (2024)

"A digital twin is a set of ***virtual information constructs*** *that* ***mimics the structure, context, and behavior*** *of a natural, engineered, or social system (or system-of-systems), is* ***dynamically updated*** *with data from its physical twin, has a* ***predictive capability****, and informs decisions that realize value."*

NASEM Foundational research gaps and future directions for digital twins, 2024

What is a Digital Twin?



DT model in sync with PT

- DT is both model and control
- DT is a "live replica" of PT

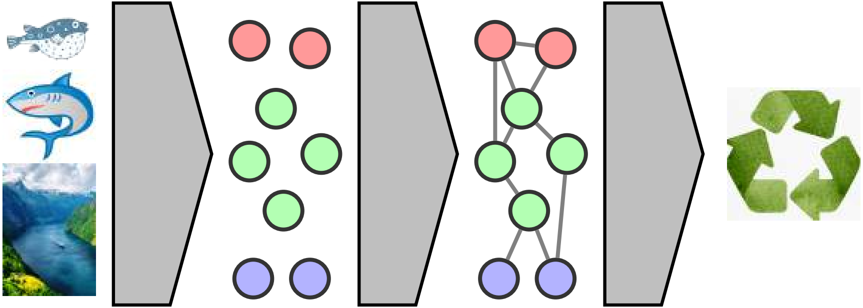
Consequences of DT as model

- Explore decision space for PT
- Lifespan of DT and PT may differ

Lifecycle Management

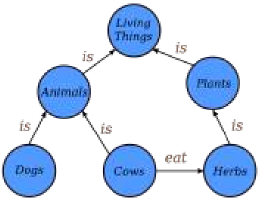
What does it mean to be a “live replica”?

- **Connects designs, requirements and software** that go into the system represented by the DT
- Evolve in tandem with **PT lifecycle stages**: design, development, operation, decommissioning, …
- Digitalisation turns this **business management problem** into a **software engineering problem**



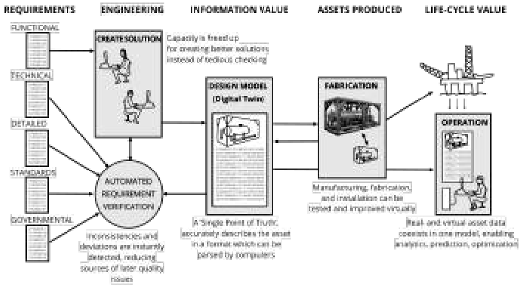
Domain Knowledge & Asset Models

[Wikipedia]



What is an asset model?

Asset models capture the knowledge of Subject Matter Experts in a framework that can be used to answer different business questions



[Readi project]

- **Asset characteristics**: configuration, design documents and simulations, standards, failure models
- **Condition data**: current state of the asset
- **Operational and environmental data**: loading, duty, rate information, corrosion rates, etc
- **Business risk and cost**: value framework, quantification of risk, costs of labour, equipment, etc

Digital Twins: Conceptual Layers

Behavioral twin
e.g., simulation framework

$\updownarrow$

Structural twin
domain knowledge, asset model, twin model

$\updownarrow$

Data:
observations, parameters, domain model

Need for different "insights":

- **Descriptive:** Insight into the past ("what happened")
- **Predictive:** Understanding the future ("what may happen")
- **Prescriptive:** Advise on possible outcomes ("what if")
- **Reactive:** Automated decision making

Programming challenges for DTs

Model-centric software

1. from business problems
   to software engineering problems
2. from software engineering problems to general programming with knowledge graphs
3. from general software to model management, federation and configuration

. . . but how do we program that?

Part 2

How can we program digital twins?

Reflection & Evolution

What is reflection?

"Reflection is the ability of a process to examine, introspect, and modify its own structure and behavior" (Wikipedia)

In particular, with respect to **external reference points**

Can we use reflection to address evolution?

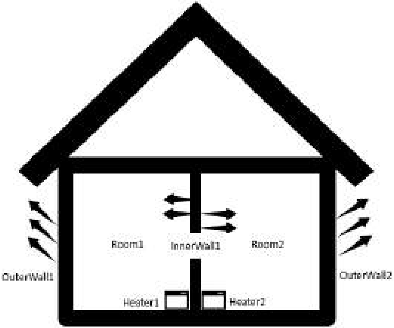
- DT needs to evolve in tandem with PT
- Reflection: tricky programming task!
- Digital thread as external reference point
- Domain knowledge as ext. reference point

Self-Adaptation & Models@Runtime

- **Domain model**: PT variability space
- **Asset model**: Current PT configuration
- **Domain of analysis**: DT variability space
- **Twin model**: Current DT configuration

Example: House Heating

[Open Simulation Platform]



Structural twin

- **Domain knowledge about houses:**
what are houses? what are rooms and connections between rooms, with corresponding simulators, etc
- **Asset model:**
instance of domain knowledge for particular house
- **Domain knowledge for analysis:**
the configuration space for behavioral twins
- **Twin model:** instance of domain knowledge for particular analysis problem

Behavioral twin

- **Digital twin:** integrates observations, monitors and orchestrates simulators
- **Twin configuration:** simulators corresponding to the different parts of the asset

[Kamburjan, Klungre, Schlatte, Tapia Tarifa, Cameron, Johnsen: *Digital Twin Reconfiguration Using Asset Models*. ISoLA 2022]

Twinning in SMOL

Programming support for twins with a behavioral and a structural layer

SMOL: Semantic Model Object Language

- SMOL is a small OO programming system which supports reflection into knowledge bases
- Runtime states in SMOL are automatically lifted into a structural model, and integrated with domain knowledge formalised using ontologies
- Ontology reasoners allow querying the KB
- Programs can use reasoners to query the KB about themselves

[Kamburjan, Klungre, Schlatte, Johnsen, Giese: *Programming and Debugging with Semantically Lifted States*. ESWC 2021]

Behavioral twins in SMOL

- SMOL can encapsulate simulators based on the FMI standard
- Using semantic reflection in SMOL, the twin configuration is automatically lifted into the structural twin

https://smolang.org

Behavioral twins in SMOL

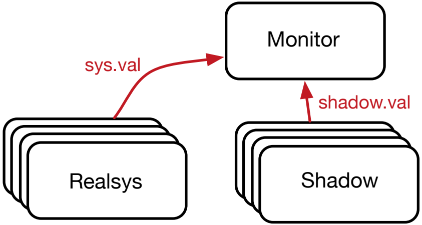
Dynamically created model instances

- Need interface to continuous models
- For example, ML models, black-box simulators (e.g., proprietary), …
- FMI is an industry standard interface to simulators
- Continuous models can be dynamically embedded in SMOL objects

```
class Room(FMO[out Int i] fmo) end
main
  FMO[in Int j, out Int i] cont = simulate("path/to/fmu", j=1, k=1);
  Room c = new Room(cont);
  cont.doStep(0.1); // FMI step function
  Int v = cont.i; cont.j = v+1; // input/output to the simulator
end
```

[Kamburjan, Johnsen: *Knowledge Structures Over Simulation Units*. ANNSIM 2022]

Example: A Self-Configuring Behavioral Twin

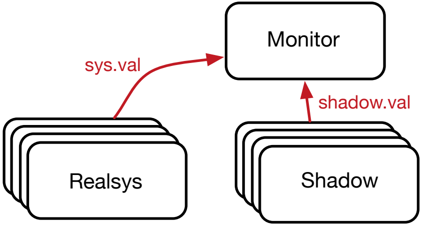


DT orchestration

- **Realsys:** FMO wrappers for system observations
- **Shadow:** FMO wrappers for the behavioral twin
- **Monitor** drift between model and system

```
FMO[out Double val] sys = simulate("Realsys.fmu");
FMO[out Double val] shadow = simulate("Sim.fmu", val=sys.val, p=1.0);
Monitor monitor = new Monitor(1.0);
monitor.run(sys, shadow);
```

Example: A Self-Configuring Behavioral Twin



Model search

- **findNewShadow** realises a model search strategy
- Many possibilities for selecting parameter values or selecting between different simulation units
- Need to organise this configuration space!

```
class Monitor(Double threshold)
 Unit run(FMO[out Double val] sys, FMO[out Double val] shadow)
  while shadow != null do
    Double last = sys.val;
    sys.doStep(1.0);  shadow.doStep(1.0); // advance time
    Double d = sys.val − shadow.val; // compute model drift
    if(d ≥ threshold) then this.findNewShadow(shadow.val, last); end
end end end
```

Connecting Runtime States to a Knowledge Graph

The SMOL interpreter implements semantic lifting, the process of generating a knowledge graph from the current program state

SMOL knowledge graph

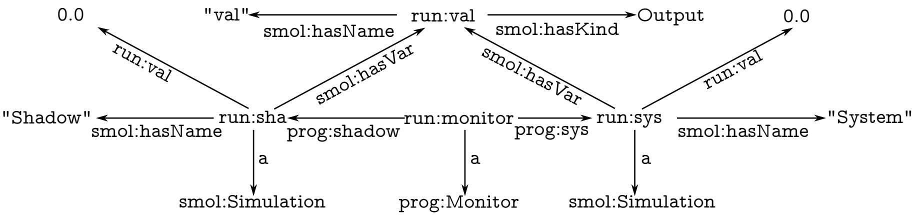
- SMOL ontology: defines the general vocabulary and basic axioms for states
- Class definitions of the SMOL program
- Knowledge generated from current runtime state (object instances, heap and stack)
- User-defined domain ontology, if supplied

Semantic lifting

- The SMOL ontology and domain ontology are given as files
- *Virtualized* heap and class table: knowledge graph built on-demand for specific query
- Type-safe language-integrated queries

[Kamburjan, Klungre, Schlatte, Johnsen, Giese: *Programming and Debugging with Semantically Lifted States*. ESWC 2021]
[Kamburjan, Klungre, Giese: *Never Mind the Semantic Gap: Modular, Lazy and Safe Loading of RDF Data*. ESWC 2022]

Semantic Lifting from SMOL Runtime States



Knowledge graph for the behavioral twin architecture

- **OWL classes** for simulators, input and output ports
- **OWL properties:** Axioms describing relations between classes

Semantic lifting of FMOs (X, ***path***, ***fmu***, ***buffer***)

- Each FMO X is an instance of smol:simulation
- X smol:hasName *name(path)*
- Each variable in *buffer* is related to an instance of run:val

Example: Programming with Reflection

SMOL programs can query the structural twin for information about its own state

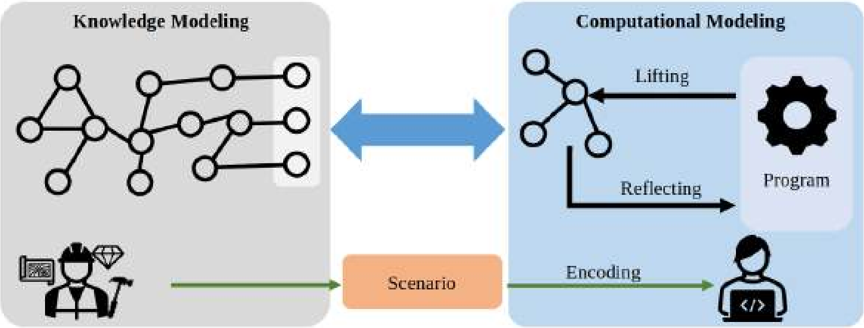
```
class Building(List<Room> rooms) ... end

class Inspector( )
 Unit inspectStreet(String street)
  List<Building> buildings := access("SELECT ?x WHERE {?x a Villa. ?x :in %street}");
  this.inspectAll(buildings);
 end
end
```

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

SMOL's Modeling Bridge

We now want to connect the SMOL program to an external asset model



- Use SMOL (and external simulators) to capture the effects of a process
- Interpret state via ontology expressing domain knowledge

[Qu, Kamburjan, Torabi, Giese: *Semantically triggered qualitative simulation of a geological process*. Appl. Comp. and Geosc. 21, 2024]

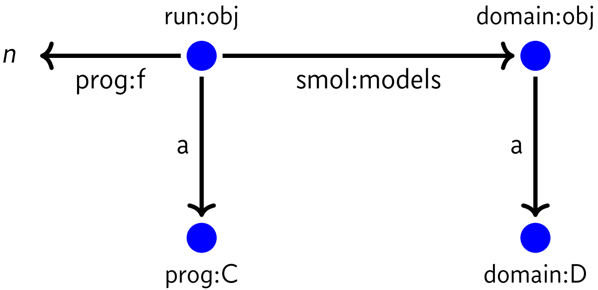
SMOL's Modeling Bridge

Bridging the gap

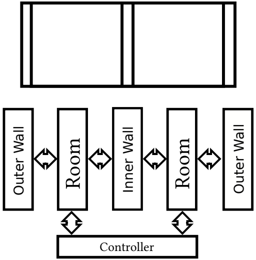
How to express what a SMOL runtime object represents in the domain?

```
class C(Int f,          ) models "a domain:D"

end
```



Twinning the House



Twinning the house

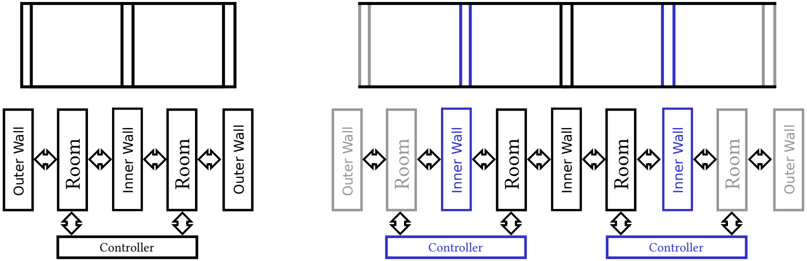
1. The asset model specifies simulators for the different physical components
2. The behavioral twin adds a controller to adjust heaters of adjacent rooms

Correctness of the behavioral twin

We can relate the structure of the asset to the structure of the behavioral twin:

- The components and structure of the asset are exactly mirrored by the twin

Structural Evolution of the Asset



Extending the house

1. New rooms are added to the house
2. Twin needs to reconfigure the simulation model and replace the controller

Structural evolution of behavioral twins

Idea: Use the structural twin to detect the *structural drift* between asset and twin as a basis for model repair of the behavioral twin

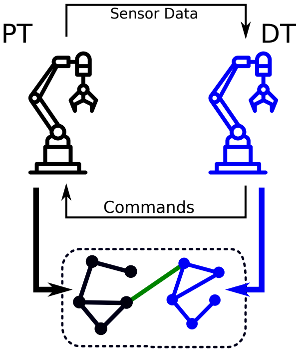
Structural Self-Adaptation

Status

- Asset model provides access to sensors from the PT,
- Asset model provides access to structure of the PT
- SMOL program can simulate model(s) of the PT

Putting it all together

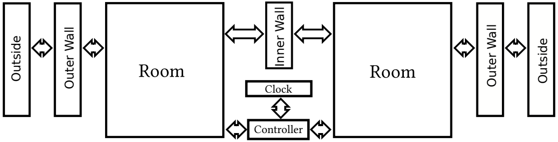
- Compare simulations to sensors
- Compare digital with physical structure
  How to formalize consistency?
- Self-adapt to changes in PT
  How to repair?



[Kamburjan, Klungre, Schlatte, Tapia Tarifa, Cameron, Johnsen: Digital Twin Reconfiguration Using Asset Models. ISoLA 2022]

DT Consistency

- A DT is **consistent** if it has the correct structure to operate on the current state of the PT
- DT uses correct models, correct configurations, adheres to current requirements
- How to formalize this in terms of reflection?



Digital Twin Consistency

- Define each consistency contraint for “correct structure” as a defect query
- A **defect query** returns a witness for the violation of some consistency constraint
- DT is considered consistent it all defect queries return an empty set

Detecting Changes in the Asset

Interacting with the structural twin

- Defect query detect changes between the asset and the simulation model
- Example: Construct list of objects from SPARQL query

```
class RoomAssert(String room, String wallLeft, String wallRight) end
  ....
  List<RoomAssert> newRooms =
   construct("SELECT ?room ?wallLeft ?wallRight WHERE
      { ?x a asset:Room;
          asset:right [asset:Wall_id ?wallRight];
          asset:left [asset:Wall_id ?wallLeft]; asset:Room_id ?room.
        FILTER NOT EXISTS {?y a prog:Room; prog:Room_id ?room.} }");
```

Note: The query relates individuals in the asset model to runtime objects

Evolving the Behavioral Twin

Identifying structural drift

- Both rooms to the left of the old house
- Both rooms to the right
- A room on either side

Reconfiguring the behavioral twin

1. **Create the new simulation elements** and insert them into the structure.
2. **Repair virtual elements** that are not reflecting elements in the asset
3. **Validate result:** Using reflection, we can check that the behavioral twin now mirrors the asset: defect queries return empty sets after repair

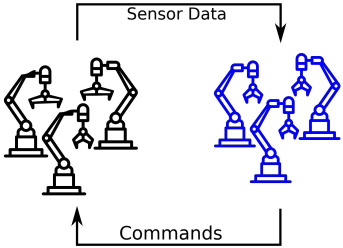
Digital Twins as Self-Adaptive Systems

Self-adaptive systems

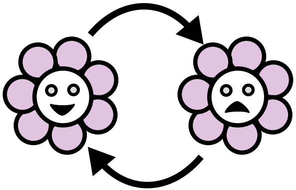
- So far: Lifting of digital twin software and use of defect queries
- How to organize the relation between digital twin and digital thread?

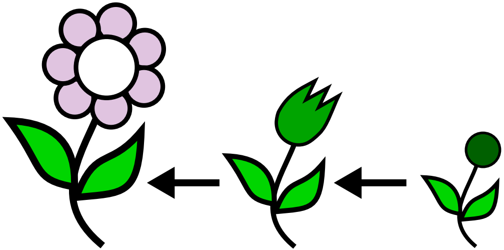
MAPE-K architecture for self-adaptation

- Split system in managing system and managed system
- **M**onitor managed systems, **A**nalyze its defects, **P**lan its repair and **E**xecute the plan based on **K**nowledge
- Where is the MAPE-K loop in digital twins?



Lifecycles and Structural Self-Adaptation





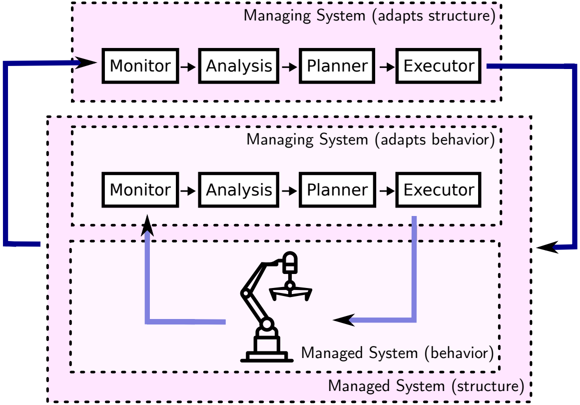
- Components of the physical twin have different lifecycle stages
- Each lifecycle stage requires a different setup, different MAPE components etc.
- May also be part of multiple lifecycles, lifecycles may interact
- Do we really need to model the whole transition system?

Operational vs. Declarative Lifecycles

- An operational lifecycle describes how to change between different stages
- A declarative lifecycle describes what it means to by at different stages

[Kamburjan, Bencomo, Tapia Tarifa, Johnsen: *Declarative Lifecycle Management in Digital Twins*. EDTconf 2024]

Digital Twins as Two-Layered Self-Adaptive Systems



- **Lifecycle stages are declarative**, defined by two predicates
- **Membership predicate:** When an asset is considered to be in a stage
- **Consistency predicate:** When an asset's assigned components are considered consistent with its stage
- **Compatibility between stages**: Compatibile stages may restrict each other's consistency (similar to cross-product)

Self-adaptation based on abduction

Find explanation about components that make DT consistent with PT at detected lifecycle stage(s)

Correctness

How to ensure quality and correctness of a DT?

Quality of a DT = Ability to reconfigure?

We have argued that correctness of a DT is its ability to adapt to changes in the PT, including *changing the correctness criteria* for the PT

Examples using KG and reflection for detection and repair of model drift

- Do we use the **correct requirement monitor** at the current lifecycle stage?
- Do we apply the **correct PT controller** at the current lifecycle stage?
- Do current lifecycle stages always result in **valid configurations**?

How to make sure that the program interactions with the KG are correct?

- **Typing:** Do the queries from the program respect the object-oriented model of SMOL?
- **Testing:** Detecting bugs in self-adaptive digital twins

Correctness of the Behavioural Twin

SMOL programs can query the structural twin to check (invariant) properties for the behavioural twin (e.g., after the model search)

Consistency of lifted state wrt domain knowledge

- **Example:** An FMO with name Shadow is of class ShadowFMO
- **Example:** The range of the relation prog.shadow is ShadowFMO

OWL

Shapes: Constraints on subgraphs of the knowledge graph

- **Example:** Every node of class prog:Monitor has a path through the properties prog:shadow and hasName to "Shadow"

SHACL

Queries: Express negative properties

- **Example:** The set of Monitor instances that have loaded a simulator FMO as a connection to the real system. (This set should be empty!)

SPARQL

Type Safety of Semantic Reflection

Types & subject reduction for semantically reflected programs

- SMOL is statically typed, … even with an untyped query language
- Given a knowledge graph, we can guarantee subject reduction for well-typed programs?

$$\frac{\mathit{answers}(Q) \subseteq \mathit{members}(C)}{\Gamma \vdash \text{List<C> l:=}\textbf{access}(Q);\ :\ \text{Unit}}$$

$$\frac{Q \subseteq \{\text{?x a prog:C.}\}}{\Gamma \vdash \text{List<C> l:=}\textbf{access}(Q);\ :\ \text{Unit}}$$

$$\frac{\Phi(Q) \sqsubseteq \text{prog:C}}{\Gamma \vdash \text{List<C> l:=}\textbf{access}(Q);\ :\ \text{Unit}}$$

$$\frac{\exists C.\ \exists \bar{y}.(\phi) \sqsubseteq^{\mathcal{T}} C \sqsubseteq^{\mathcal{T}} \mathit{Class}_{\mathrm{T}'} \quad \Gamma \vdash \text{l} : \text{List<T}'\text{>} \quad \Gamma \vdash \text{e}_i : \mathrm{T}_i}{\Gamma \vdash^{\mathcal{T}}_{\text{er}} \text{l:=}\textbf{access}(\exists \bar{y}.\ \phi, \text{e}_1, \ldots, \text{e}_n)\ :\ \text{Unit}}$$

Queries

- Query containment becomes our subtyping relation
- Subtleties wrt entailment regimes in knowledge graphs
- Complexity tractable if query translates into DL concept

[Kamburjan, Kostylev: *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL 2021]

Can We Test Software Interacting with Knowledge Graphs?

Testing Knowledge Graphs Applications

- What exactly to test? Unit testing? Integration testing?
- How to get a test oracle?
- Main challenge: Knowledge graphs are highly structured inputs

Testing Knowledge Graphs Applications

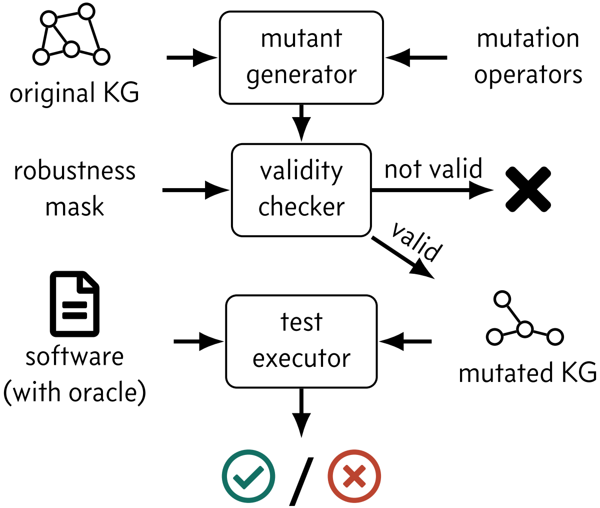
- Generating random triples is easy
- Generating triples adhering to an ontology requires reasoning
- Mutating triples also requires reasoning
- Mutating single triples either obviously breaks system or changes too little

[John, Johnsen, Kamburjan: *Mutation-Based Integration Testing of Knowledge Graph Applications*, ISSRE 2024]

Mutation Testing of Knowledge Graph Applications

Approach

- **Main idea 1**: Domain-specific mutations change bigger parts of KG
- **Main idea 2:** Robustness mask to specify where mutations are allowed
- **Main idea 3:** Monitoring queries as testing oracles



- Mutation testing on ontologies and knowledge graphs largely unexplored
- Existing approaches change single triples, but these contain little information
- Domain-specific operations add or change whole subgraphs

Example: GreenhouseDT

GreenhouseDT: System Overview

[Kamburjan, Sieve, Baramashetru, Amato, Barmina, Occhipinti, Johnsen: *GreenhouseDT: An Exemplar for Digital Twins*, SEAMS 2024]

GreenhouseDT: System description

- **Physical twin:** greenhouse, sensors and actuators
- **Digital twin:** extensible software architecture that realizes behavioral self-adapatation (adaptive control) and architectural self-adaptation

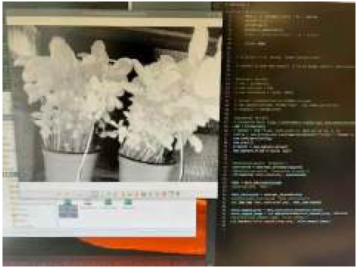
- **Simulation Model:** program providing operations for self-adaptation and control, reflecting the asset's architecture (e.g., plants, actuators and their connections)
- **Driver:** triggers the self-adaptation and control loops of the simulation model, relays decisions to actuators
- **Knowledge Graph:** the asset model — a formal description of the physical twin
- **Time-series database:** interface to the sensors

Self-adaptation in GreenhouseDT

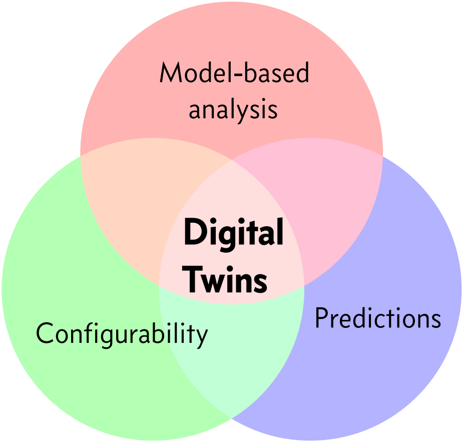
- **Behavioral self-adaptation:** adjust controller of water pump to reach goal in asset model (e.g., humidity level in the pot)
- **Architectural self-adaptation:** change monitor & controller to reflect changed goal (e.g., change of season) and plant health

Plant Health Monitoring

- **Physical twin:** infrared camera to the physical twin
- **DT Infrastructure:** Add NVDI values to database schema and health status threshold values to the asset model
- **Simulation Model:** Uses defect queries to adapt to NVDI thresholds



Conclusion



Digital twins: summary

- **From model-driven to model-centric engineering**
  - May change how programs are built in the future!
- **Range of application domains:**
  - Huge industry interest
  - Cyber-physical systems in the large
- **Range of analysis techniques needed:**
  - Descriptive, predictive, prescriptive
- **State of practice today:**
  - Ad hoc solutions: brittle, inflexible, monolitihic
  - Lack of established software architectures

Contributors to this Research

Shout outs to:

The great team who contributed to the work presented in this talk

- Eduard Kamburjan
- Silvia Lizeth Tapia Tarifa
- Andrea Pferscher
- Riccardo Sieve
- Nelly Bencomo
- Rudi Schlatte
- Tobias John
- Martin Giese
- Yuanwei Qu
- Egor Kostylev
- Vidar Klungre
- David Cameron

Collaborators and master students on the research projects

- A Digital Twin of the Oslo Fjord
- A Digital Twin for Pandemic Prediction
- Digital Arctic Twins
- BedreFlyt
- Sirius
- NebulOuS