The Semantically Reflected Digital Twin

Einar Broch Johnsen
Eduard Kamburjan
Silvia Lizeth Tapia Tarifa

Incorporates material from
Martin G. Skjæveland, Dimitris Kyritsis, Qianhang Lyu
Supported by EU Project SM4RTENANCE
University of Oslo, Norway

`MODELS'24 Tutorial, Linz, 22.09.24`

Digital Twins — The Hype

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

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
2. Companies increasingly create digital twins of their physical assets

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
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**

NASA's definition of a DT

*"an **integrated multi-physics, multi-scale**, probabilistic simulation of a vehicle or system that uses the **best available** physical models, sensor updates, fleet history, etc., to **mirror the life of its flying twin**. It is **ultra-realistic** and may consider one or more important and interdependent vehicle systems"*

NASA Modeling, Simulation, Information Tech. & Processing Roadmap, 2010

Digital Twins and Twinning

A DT connects an asset with

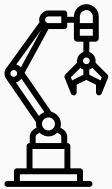
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.

Digital Twins and Twinning

A DT connects an asset with

1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.

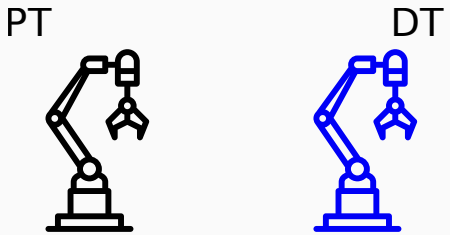
PT



Digital Twins and Twinning

A DT connects an asset with

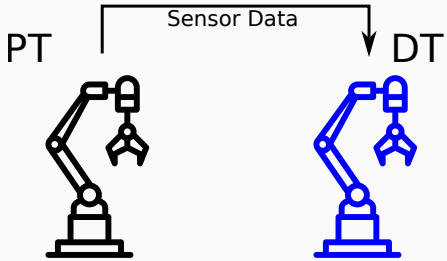
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.



Digital Twins and Twinning

A DT connects an asset with

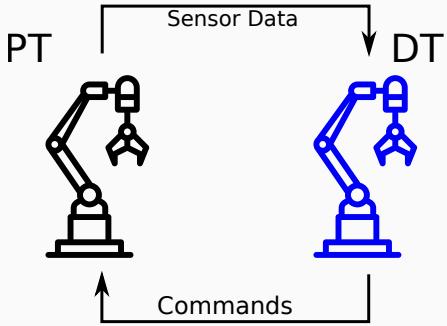
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.



Digital Twins and Twinning

A DT connects an asset with

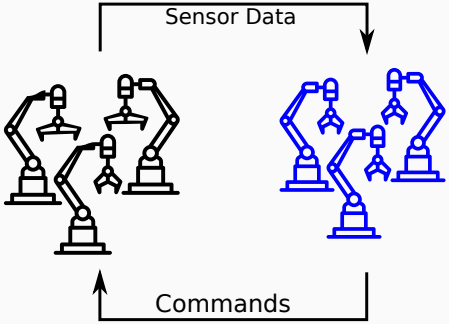
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.



Digital Twins and Twinning

A DT connects an asset with

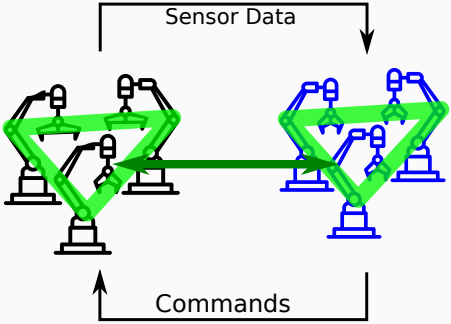
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.



Digital Twins and Twinning

A DT connects an asset with

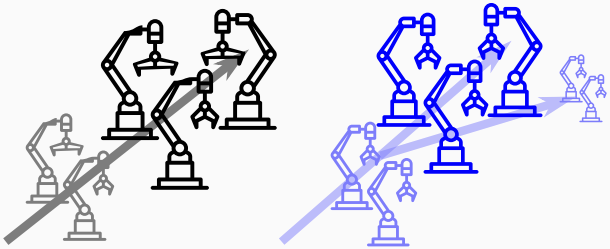
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.



Digital Twins and Twinning

A DT connects an asset with

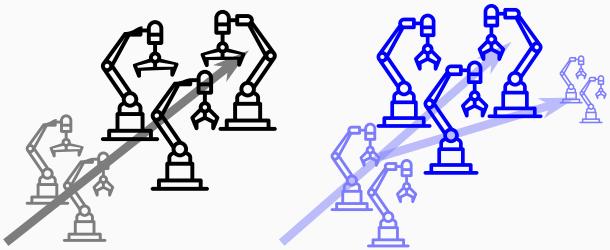
1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.



Digital Twins and Twinning

A DT connects an asset with

1. its own (simulation) models using data streams and commands, and
2. its historical and design data/models.

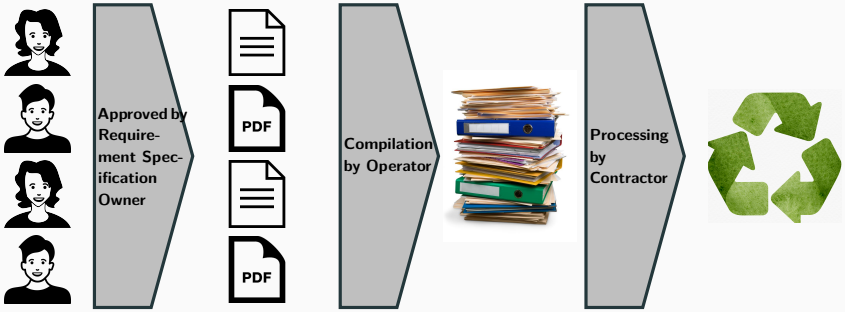


- Digital twin must evolve in tandem with the asset
- Digital twin must consider domain knowledge during design and operations

Lifecycle Management

Digital thread: The digital twin evolves in tandem with the asset

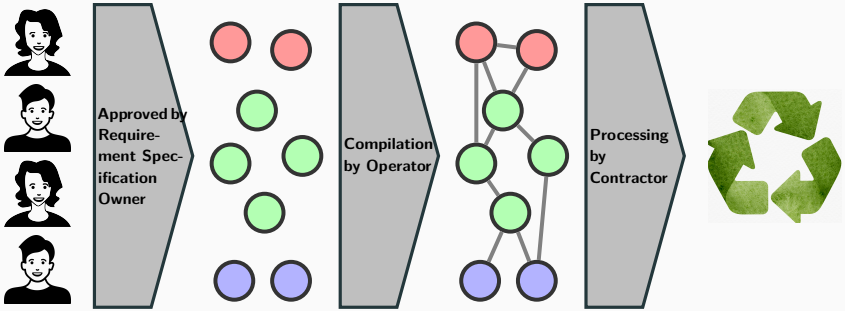
- Connects designs, models, logs, requirements and software that go into the system represented by the DT
- Connects system phases to the DT: design, operation, maintenance, ...
- Enormous **business management problem**



Lifecycle Management

Digital thread: The digital twin evolves in tandem with the asset

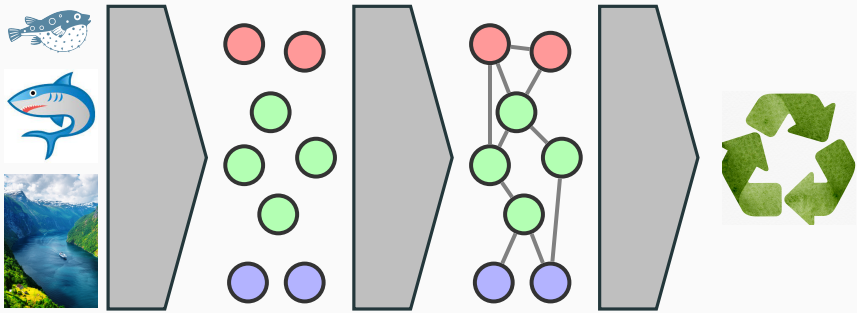
- Connects designs, models, logs, requirements and software that go into the system represented by the DT
- Connects system phases to the DT: design, operation, maintenance, ...
- Digitalisation turns this into a **software engineering problem**



Lifecycle Management

Digital thread: The digital twin evolves in tandem with the asset

- Connects designs, models, logs, requirements and software that go into the system represented by the DT
- Connects system phases to the DT: **Similar lifecycles for natural systems**
- Digitalisation turns this into a **software engineering problem**



Domain Knowledge & Asset Models

What is an asset model?

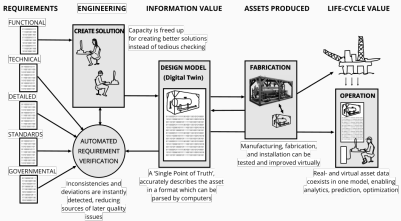
Asset models capture the knowledge of Subject Matter Experts in a framework that can be used to answer different business questions.

Domain Knowledge & Asset Models

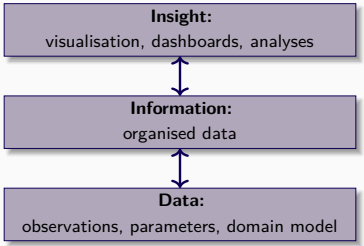
What is an asset model?

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

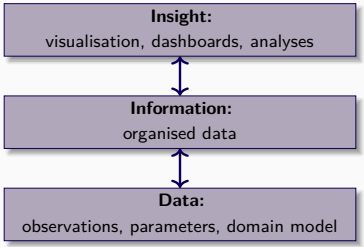
- **Asset characteristics**: configuration, design documents and simulations, standards, failure models
- **Condition data**: current state of the asset
- **Operational and environmental data**: loading, duty, rate information, corrosion rates, etc
- **Business risk and cost**: value framework, quantification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



Digital Twins: Conceptual Layers



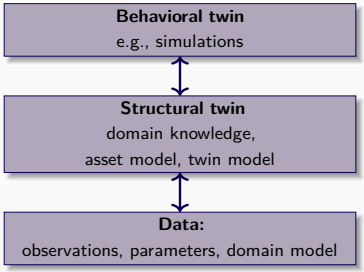
Digital Twins: Conceptual Layers



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

Digital Twins: Conceptual Layers



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?

Tutorial Outline

- **Part I** Semantic Reflection

  How to connect software with the digital thread?

- **Part II** Self-Adaptation in Digital Twins

  How to use asset models for structural self-adaptation?

- **Part III** Correctness

  How to ensure type safety, functional correctness, and test digital twins?

- **Part IV** Asset Information Modeling

  How to develop asset models?

Semantic Reflection in Programs

The need for reflection

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.

- Digital twin must be able to reflect if it evolves in tandem
- Notoriously difficult programming task in itself
- How to use digital thread as external reference point?
- How to use external knowledge graphs as reference point?

A very short primer on knowledge graphs

Triple-Based Knowledge Representation

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

A very short primer on 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.

A very short primer on 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.

RDF:

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

A very short primer on 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.

RDF:

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

OWL:

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

$\exists$`hasChild`. $\exists$`hasChild`. `Person` $\sqsubseteq$ `GrandParent`

A very short primer on 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.

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

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

$\exists$`hasChild`. $\exists$`hasChild`. `Person` $\sqsubseteq$ `GrandParent`

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

Assets as Knowledge Graphs

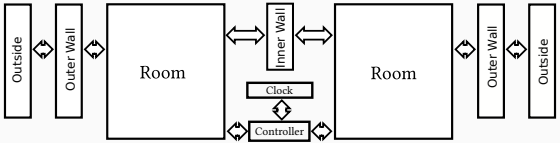
Asset Models

Asset models contain the current, past and designed structure of a facility, plus general knowledge for it. Aim: Use graph-based asset models to manage engineering lifecycle.

Assets as Knowledge Graphs

Asset Models

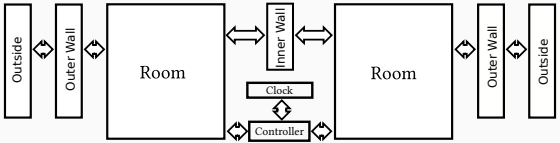
Asset models contain the current, past and designed structure of a facility, plus general knowledge for it. Aim: Use graph-based asset models to manage engineering lifecycle.



Assets as Knowledge Graphs

Asset Models

Asset models contain the current, past and designed structure of a facility, plus general knowledge for it. Aim: Use graph-based asset models to manage engineering lifecycle.

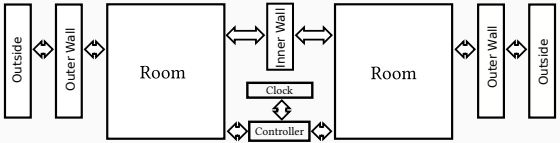



```
ast:heater1 a ast:Heater. ast:heater1 ast:in ast:room1.
ast:heater2 a ast:Heater. ast:heater2 ast:in ast:room2.
ast:heater1 ast:id 13. ast:heater2 ast:id 12.
ast:room1 ast:leftOf ast:room2.
```

Assets as Knowledge Graphs

Asset Models

Asset models contain the current, past and designed structure of a facility, plus general knowledge for it. Aim: Use graph-based asset models to manage engineering lifecycle.

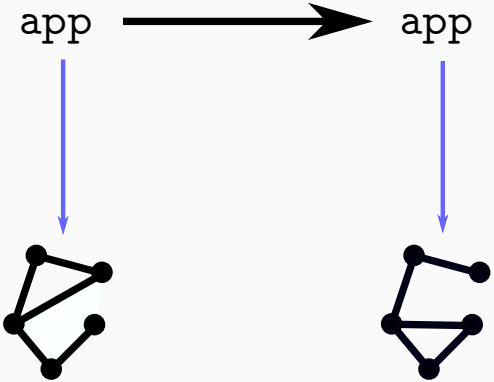



```
ast:heater1 a ast:Heater. ast:heater1 ast:in ast:room1.
ast:heater2 a ast:Heater. ast:heater2 ast:in ast:room2.
ast:heater1 ast:id 13. ast:heater2 ast:id 12.
ast:room1 ast:leftOf ast:room2.
```

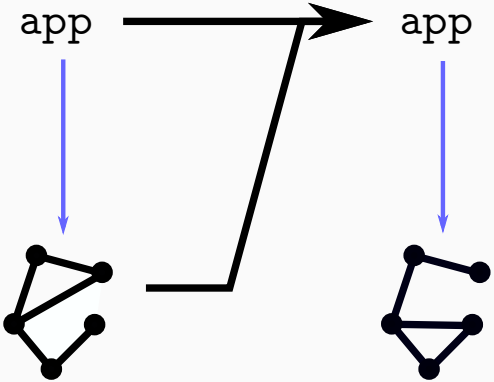
htLeftOf **subPropertyOf** `ast:in` **o** `ast:leftOf` **o inverse**(`ast:in`)

Semantically Lifted Programs

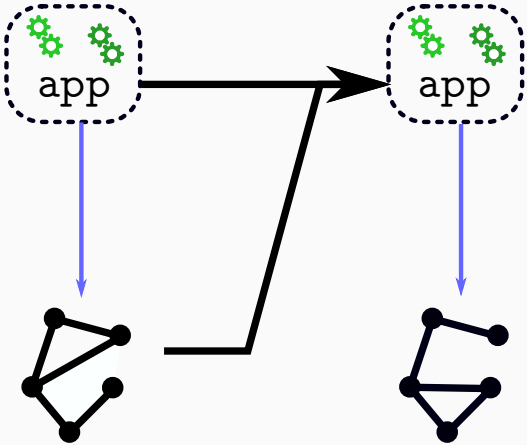
Semantically Lifted Programs



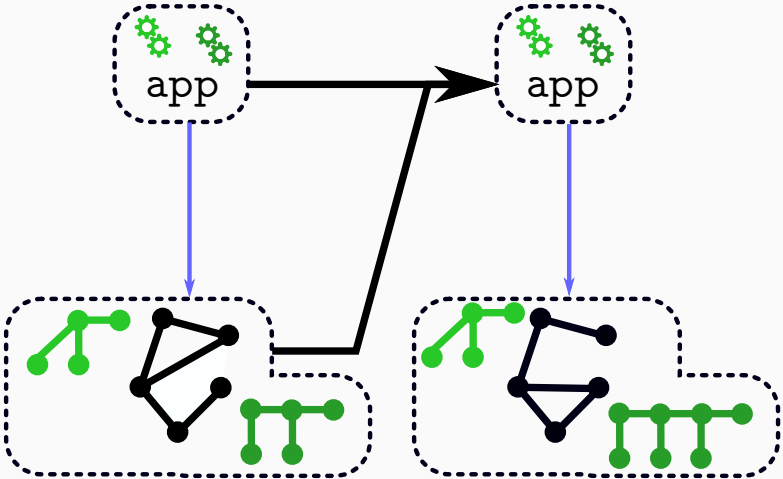
Semantically Lifted Programs



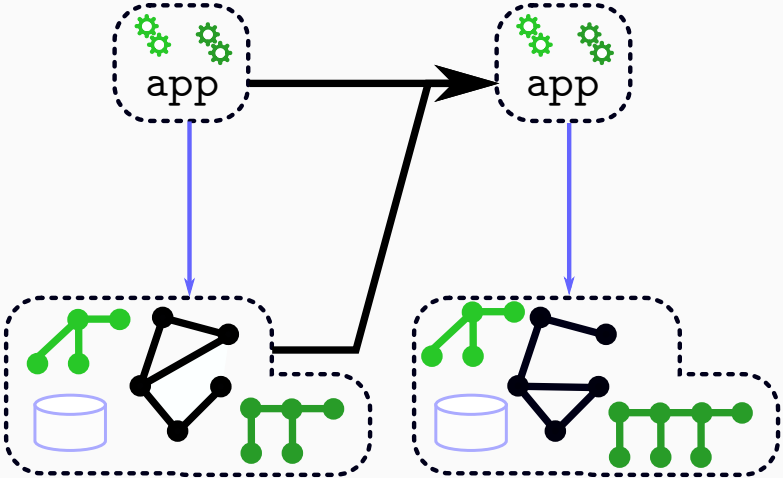
Semantically Lifted Programs



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 the program to operate on the KG. Implemented in SMOL, library for JVM available.

```
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 the program to operate on the KG. Implemented in SMOL, library for JVM available.

```
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. ...
```

[Kamburjan 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

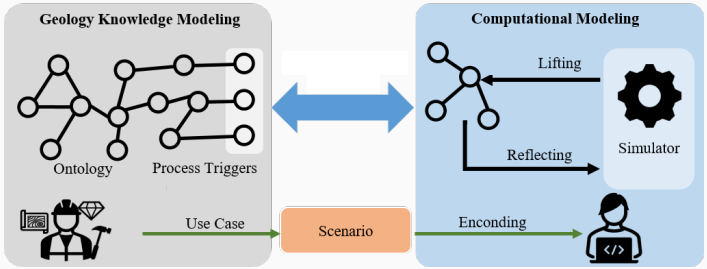
```
1 class Building(List<Room> rooms) ... end
2 class Inspector(List<Building> buildings)
3  Unit inspectStreet(String street)
4   List<Building> l := access("SELECT ?x WHERE {?x a Villa. ?x :in %street}");
5   this.inspectAll(l);
6  end
7 end
```

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 et al., *Semantically triggered qualitative simulation of a geological process*, Appl. Comp. and Geosc., 2024]

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
```

SMOL's Modeling Bridge

Bridging the gap

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

SMOL's Modeling Bridge

Bridging the gap

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

```
class C(                    )

end
```

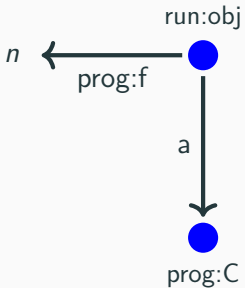
SMOL's Modeling Bridge

Bridging the gap

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

```
class C(Int f,                )

end
```



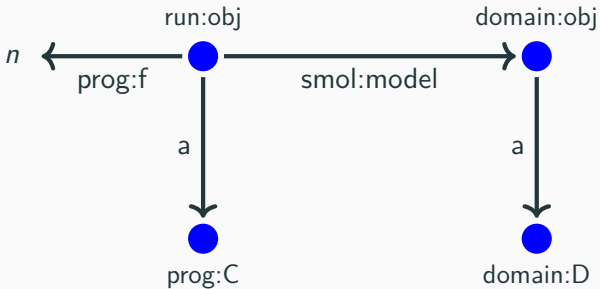
SMOL's Modeling Bridge

Bridging the gap

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

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

end
```



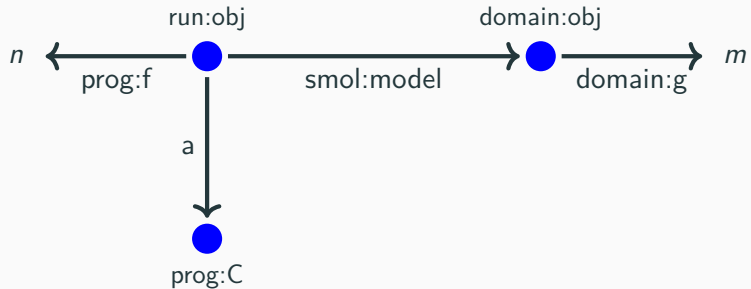
SMOL's Modeling Bridge

Bridging the gap

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

```
class C(Int f, domain Int g )

end
```



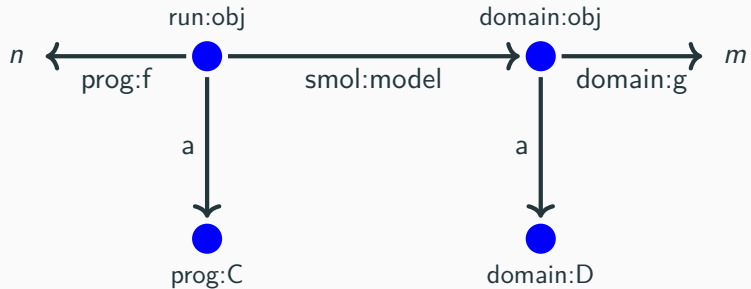
SMOL's Modeling Bridge

Bridging the gap

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

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

end
```



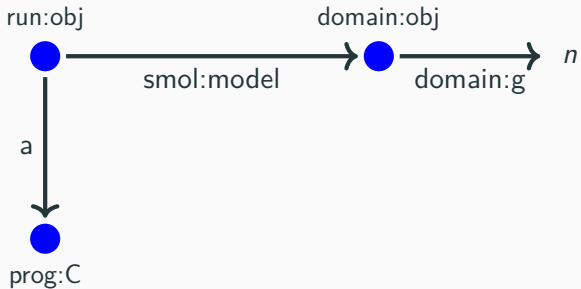
SMOL's Modeling Bridge

Bridging the gap

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

```
class C(hidden Int f,        ) models "domain:g %f"

end
```

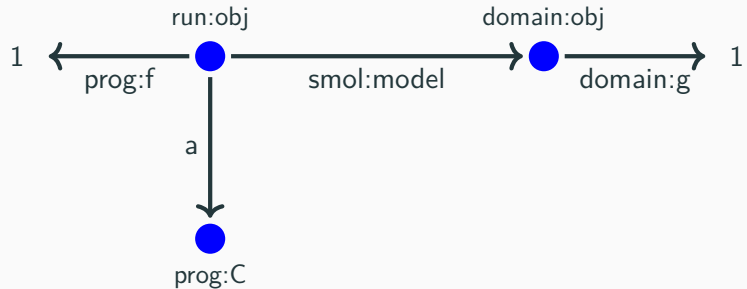


SMOL's Modeling Bridge

Bridging the gap

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

```
class C(Int f,                ) models (this.f > 0) "domain:g %f"
                                  models "a domain:D"
end
```

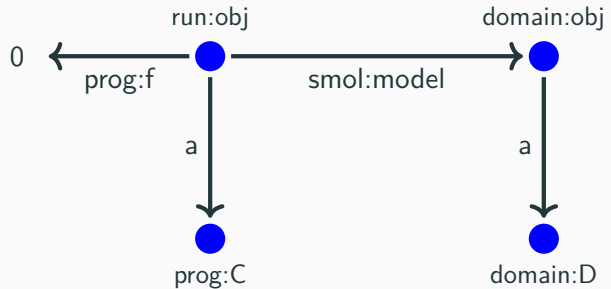


SMOL's Modeling Bridge

Bridging the gap

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

```
class C(Int f,                ) models (this.f > 0) "domain:g %f"
                                  models "a domain:D"
end
```



Semantic Reflection: Reasoning about oneself – GeoSimulator

```
1 List<ShaleUnit> fs =
2 member(smol:model some (participates_in some 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 ($2ma$ steps)

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

Self-Adaptation in Digital Twins

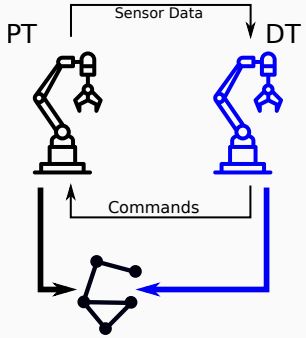
Structural Self-Adaptation

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

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

Structural Self-Adaptation

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



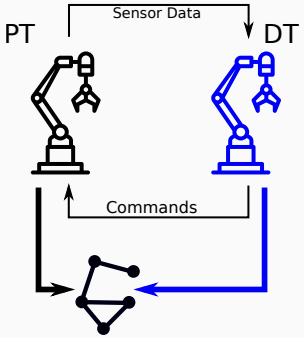
[Kamburjan et al., *Digital Twin Reconfiguration Using Asset Models*, ISoLA'22]

Structural Self-Adaptation

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

Putting it all together

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



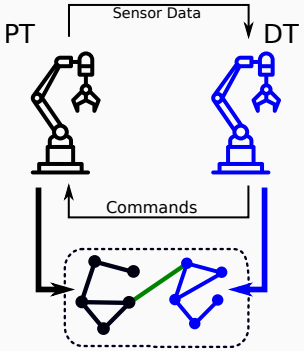
[Kamburjan et al., *Digital Twin Reconfiguration Using Asset Models*, ISoLA'22]

Structural Self-Adaptation

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

Putting it all together

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



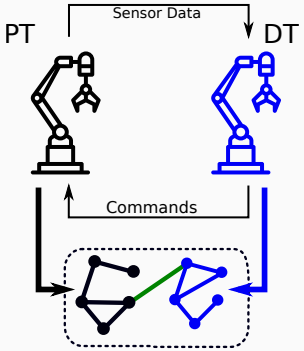
[Kamburjan et al., *Digital Twin Reconfiguration Using Asset Models*, ISoLA'22]

Structural Self-Adaptation

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

Putting it all together

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



[Kamburjan et al., *Digital Twin Reconfiguration Using Asset Models*, ISoLA'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. Tight integration as FMOs, type of FMO directly checked against FMI model description

```
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

SMOL with FMOs

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

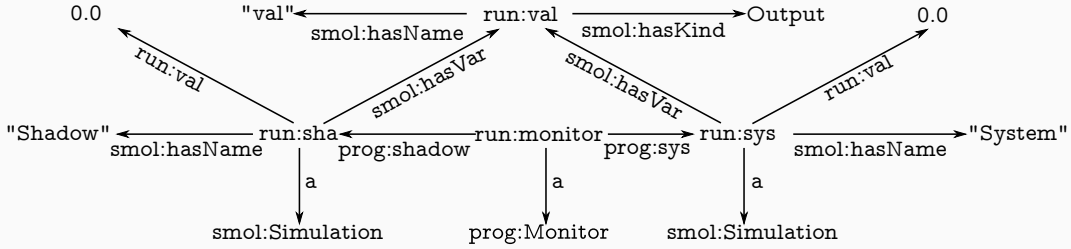
```
1 class Monitor(FMO[out Double val] sys,
2               FMO[out Double val] shadow)
```

SMOL and FMI

SMOL with FMOs

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

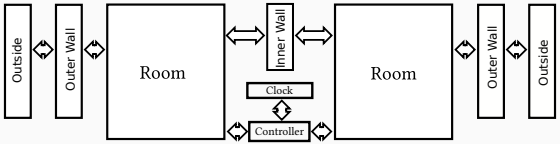
```
1 class Monitor(FMO[out Double val] sys,
2               FMO[out Double val] shadow)
```



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

Defect Queries

- A digital twin is consistent if it has the correct structure to operate on the current state of the physical twin
- Uses correct models, correct configurations, adheres to current requirements
- How to formalize this in terms of reflection?



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

Example: Adding a New Room

- Consistency constraint: for each room in the house, there is a `Room` object in the DT
- Get all (asset) rooms and their neighboring walls
- Remove all (twinned) rooms with the same id
- 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.} }");
```

Lifting Software Architectures

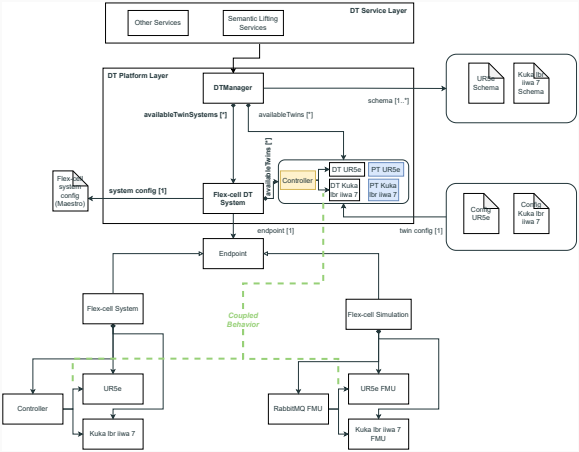
Beyond Programs

- 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

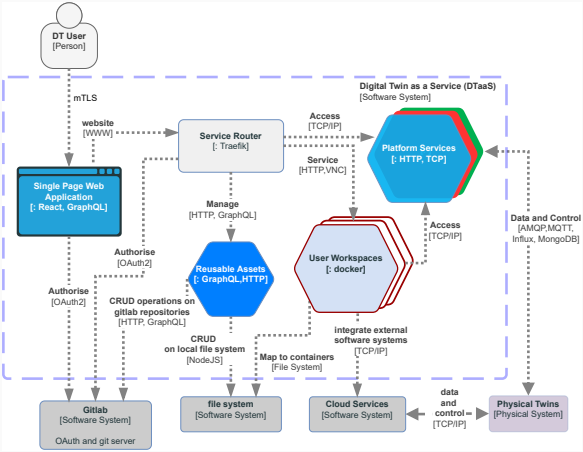


[Gil, Kamburjan, Talasila, Larsen, *An Architecture for Coupled Digital Twins with Semantic Lifting*, u.S.]

Lifting Software Architectures

Beyond Programs

- Lifting larger programs does not scale up
- Instead: Software architecture to lift only components
- Current work on DTaaS microservice platform

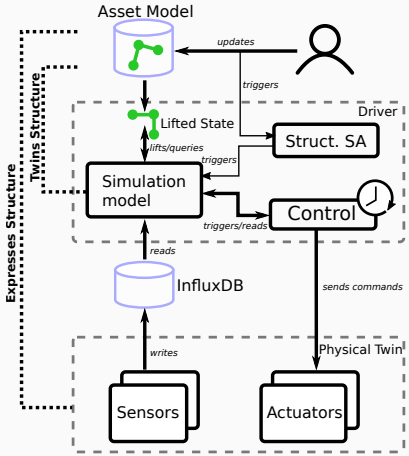


[Talasila, Kamburjan et al., *Digital Twin as a Service (DTaaS): A Platform for Digital Twin Developers and Users*, IEEE SWC'23]

Lifting Software Architectures

Beyond Programs

- Lifting larger programs does not scale up
- Instead: Software architecture to lift only components
- Current work on DTaaS microservice platform
- Digital Twin lab with Greenhouse at UiO

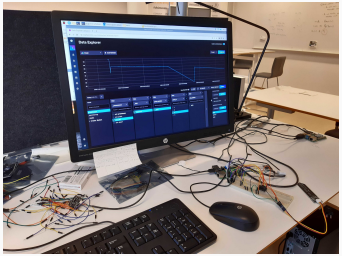


[Kamburjan et al., *GreenhouseDT: An Exemplar for Digital Twins*, SEAMS'24]

Lifting Software Architectures

Beyond Programs

- Lifting larger programs does not scale up
- Instead: Software architecture to lift only components
- Current work on DTaaS microservice platform
- Digital Twin lab with Greenhouse at UiO



[Kamburjan et al., *GreenhouseDT: An Exemplar for Digital Twins*, SEAMS'24]

Beyond architectures: 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?

Beyond architectures: 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

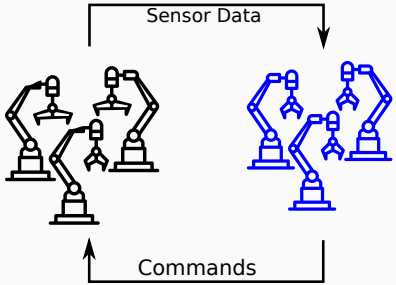
Standard architecture in self-adaptation

- 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?

Beyond architectures: Digital Twins as Self-Adaptive Systems

Self-adaptive systems

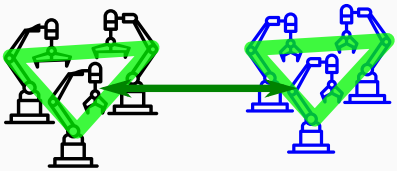
- So far: Lifting of digital twin software and use of defect queries
- How to organize the relation between digital twin and digital thread?



Beyond architectures: Digital Twins as Self-Adaptive Systems

Self-adaptive systems

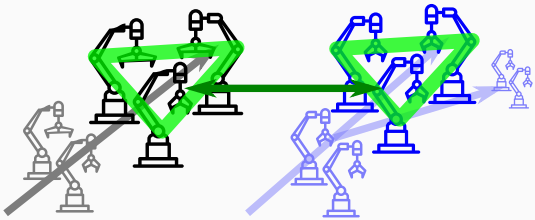
- So far: Lifting of digital twin software and use of defect queries
- How to organize the relation between digital twin and digital thread?



Beyond architectures: Digital Twins as Self-Adaptive Systems

Self-adaptive systems

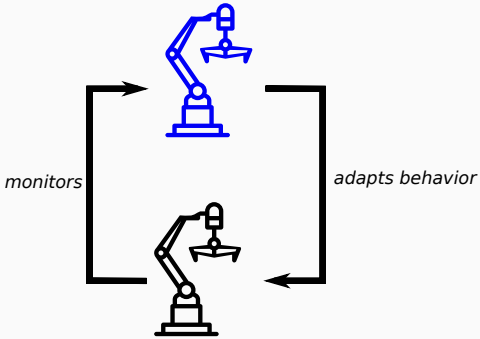
- So far: Lifting of digital twin software and use of defect queries
- How to organize the relation between digital twin and digital thread?



Beyond architectures: Digital Twins as Self-Adaptive Systems

Self-adaptive systems

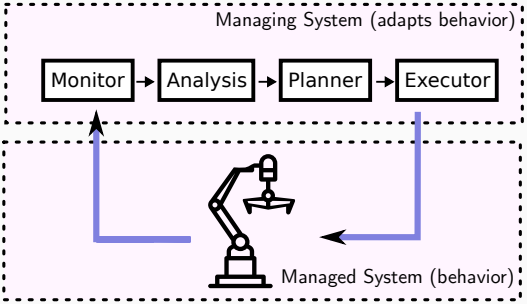
- So far: Lifting of digital twin software and use of defect queries
- How to organize the relation between digital twin and digital thread?



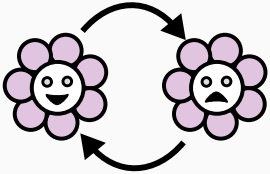
Beyond architectures: Digital Twins as Self-Adaptive Systems

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?

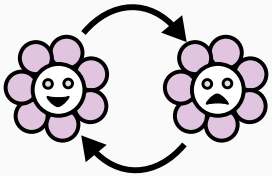


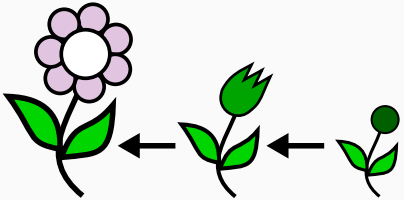
Lifecycles and Structural Self-Adaptation



- Components of the physical twin have different lifecycle stages
- Each lifecycle stage requires a different setup, different MAPE components etc.

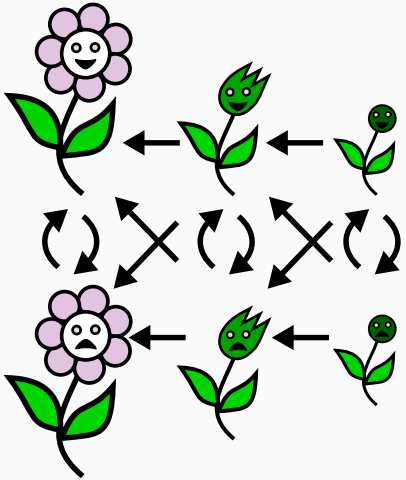
Lifecycles and Structural Self-Adaptation





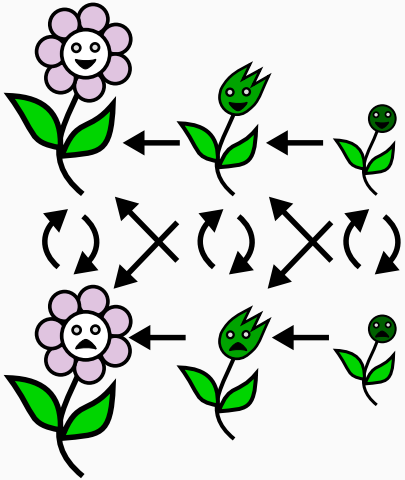
- Components of the physical twin have different lifecycle stages
- Each lifecycle stage requires a different setup, different MAPE components etc.
- May also be part of multiple lifecycles, lifecycles may interact

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?

Lifecycles and Structural Self-Adaptation

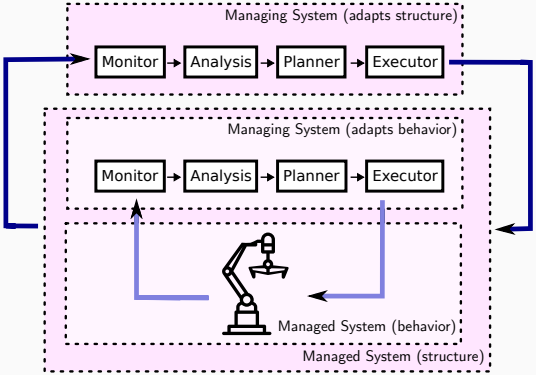


- Components of the physical twin have different lifecycle stages
- Each lifecycle stage requires a different setup, different MAPE components etc.

Operational vs. Declarative Lifecycles

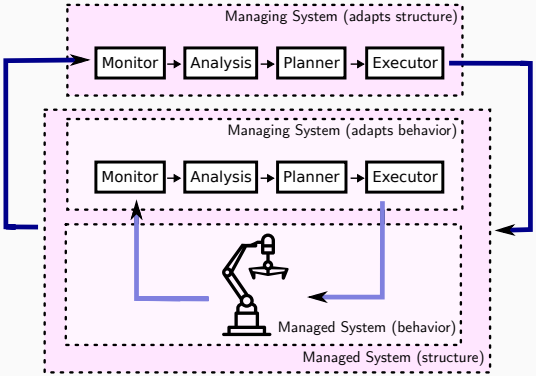
- An operational lifecycle describes how to change between stages
- A declarative lifecycle describes what it means to by at a stage

Digital Twins as Two-Layered Self-Adaptive Systems



- Second layer of self-adaptation
- Monitors the *structure* of the level-1 system
- Does also consider the state of the PT
- E.g., given a sick plant, do I have the right components to monitor its specific health requirements?

Digital Twins as Two-Layered Self-Adaptive Systems



- Lifecycle stages are declarative, with two elements as their definition
- *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
- Self-adaptation is generic: Abduct an explanation with which components as asset would be consistent with its detected stage

Declarative Lifecycle Stages

Definition (Stage)

Let $\mathcal{A}$ be an asset class. Let $\mathcal{C}$ be a set of component classes.

$$\mathrm{D}_{\mathcal{A},\mathcal{C}} = \langle \mathit{member}, \mathit{consistent} \rangle$$

- $\mathit{member} \subseteq \mathcal{A}$ are the target assets
- $\mathit{consistent} \subseteq \mathit{member} \times 2^{\mathcal{C}}$ are the required components

$$\begin{aligned} \mathrm{D}_{\mathrm{Sick}} &= \{\mathit{member}_{\mathrm{Sick}}, \mathit{consistent}_{\mathrm{Sick}}\} \\ \mathit{member}_{\mathrm{Sick}} &= \{a \mid \mathrm{nvdi}(a) \leq 0.5\} \\ \mathit{consistent}_{\mathrm{Sick}} &= \{(a, X) \mid a \in \mathit{member}_{\mathrm{Sick}}, \mathrm{analyzer}^{\leq 5}_{\mathit{moisture}}(a) \in X\} \end{aligned}$$

Lifecycles

Definition (Lifecycle)

Let $\mathcal{A}$ be an asset class and I an index set. A *lifecycle* $\mathsf{L}_{\mathcal{A}}$ for $\mathcal{A}$ is a set of declarative stages $\left(\mathsf{D}_{\mathcal{A},\mathcal{C},i}\right)_{i\in I}$ such that every asset from $\mathcal{A}$ is in exactly one stage:
(1) $\mathcal{A} = \bigcup_{i\in I} \mathit{member}_{\mathsf{D}_{\mathcal{A},\mathcal{C},i}}$ (2) $\forall i,j \in I.\ i \neq j \Rightarrow \mathit{member}_{\mathsf{D}_{\mathcal{A},\mathcal{C},i}} \cap \mathit{member}_{\mathsf{D}_{\mathcal{A},\mathcal{C},j}} = \emptyset$

$$
\begin{aligned}
\mathsf{D}_{\mathsf{Healthy}} &= \{\mathit{member}_{\mathsf{Healthy}}, \mathit{consistent}_{\mathsf{Healthy}}\} \\
\mathit{member}_{\mathsf{Healthy}} &= \{a \mid \mathsf{nvdi}(a) > 0.5\} \\
\mathit{consistent}_{\mathsf{Healthy}} &= \{(a, X) \mid a \in \mathit{member}_{\mathsf{Healthy}}, \mathsf{analyzer}^{\leq 10}_{\mathit{moisture}}(a) \in X\}
\end{aligned}
$$

Compatible Lifecycles

Definition (Compatibility)

The stages D_1 and D_2 are *compatible* if, for all $a \in member_{D_1} \cap member_{D_2}$ there is some $X \subseteq \overline{\mathcal{C}}$ such that $(a, X) \in consistent_{D_1}$ and $(a, X) \in consistent_{D_2}$

- Two lifecycles are compatible if all their stages are compatible
- Compatible stages may restrict each other's consistency, but not make it impossible
- Simple composition, akin to cross-product

Algorithm (simplified)

Definition (Abduction-Based Self-adaptation)

For one asset a, with one lifecycle.

1. Retrieve assigned components X (**Monitor**)
2. Check if $a \in \mathit{member}_{\mathrm{D}} \wedge (a, X) \notin \mathit{consistent}_{\mathrm{D}}$ (**Analyze**)
3. If so, abduce for which X', we have $(a, X') \in \mathit{consistent}_{\mathrm{D}}$ (**Plan**)
4. Remove components in $X \setminus X'$ (**Execute**)
5. Add components in $X' \setminus X$ (**Execute**)

- Require logical representation of asset and component information

Example

- New sensor value indicates that plant P is sick

$$\begin{aligned} &\text{nvdi}(\mathtt{P}) \doteq 0.4, && \mathtt{P} \in \mathit{member}_{\text{Sick}}, \\ &\text{analyzer}^{\leq 10}_{\mathit{moisture}}(\mathtt{P}) \in X, && \mathtt{P} \notin \mathit{consistent}_{\text{Sick}} \end{aligned}$$

- Abduce solution

$$\text{analyzer}^{\leq 5}_{\mathit{moisture}}(\mathtt{P}) \in X$$

- Generate and execute plan

$$\begin{aligned} &\text{nvdi}(\mathtt{P}) \doteq 0.4, && \mathtt{P} \in \mathit{member}_{\text{Sick}}, \\ &\text{analyzer}^{\leq 5}_{\mathit{moisture}}(\mathtt{P}) \in X, && \mathtt{P} \in \mathit{consistent}_{\text{Sick}} \end{aligned}$$

Correctness

Maintenance and Quality Measures

- How to ensure quality of the digital twin?
- Architectures give guidelines, can be used to analyze/generate code
- Main challenge: How to make sure that the interactions with the KG are correct?

Typing Do the queries of my program respect the object-oriented modeling?

Testing Detecting bugs in self-adaptive digital twins

Formal Specification Using ontologies to specify contracts

Correctness – Typing

Type Safety and Interfaces

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

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

Is this type safe?

Type Safety and Interfaces

```
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]
```

Is this type safe?

- Depends on the ontology – it is safe if every villa is a building
- Requires reasoning, e.g., about the domain of `rooms`

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, . . .

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, ... even with an untyped query language

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, ... even with an untyped query language
- We can guarantee safe query access if ontology $\mathcal{K}$ is known

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, ... even with an untyped query language
- We can guarantee safe query access if ontology $\mathcal{K}$ is known

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

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, ... even with an untyped query language
- We can guarantee safe query access if ontology $\mathcal{K}$ is known

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

Queries

- Query containment (wrt. entailment) becomes our subtyping relation
- (More) tractable if query translates into DL concept

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, ... even with an untyped query language
- We can guarantee safe query access if ontology $\mathcal{K}$ is known

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

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

Queries

- Query containment (wrt. entailment) becomes our subtyping relation
- (More) tractable if query translates into DL concept

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Type Safety for Semantic Reflection

Types & subject reduction

- SMOL is statically typed, ... even with an untyped query language
- We can guarantee safe query access if ontology $\mathcal{K}$ is known

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

Queries

- Query containment (wrt. entailment) becomes our subtyping relation
- (More) tractable if query translates into DL concept

[Kamburjan and Kostylev, *Type Checking Semantically Lifted Programs via Query Containment under Entailment Regimes*, DL'21]

Correctness – Testing

Testing

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

The trouble with knowledge graphs

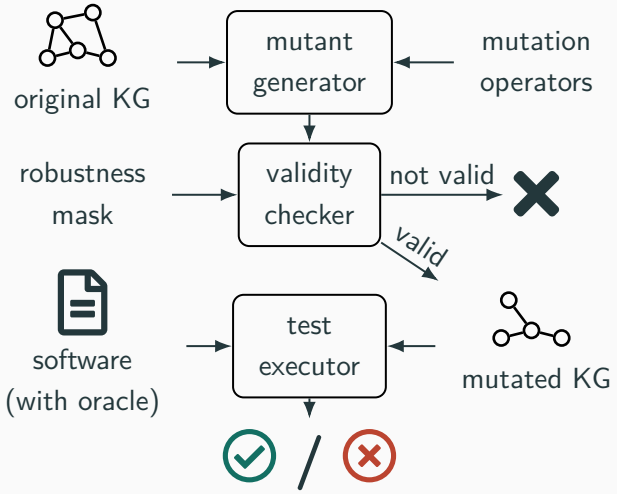
- Generating random triples is easy
- Generating triples adhering to an ontology requires reasoning
- Mutating triples also requires reasoning
- Mutating single triples either obviously breaks system or changes too little

[John, Johnsen, and Kamburjan, *Mutation-Based Integration Testing of Knowledge Graph Applications*, ISSRE'24]

Testing

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



Testing - Domain-specific mutation operators

- Mutation testing on ontologies and knowledge graphs largely unexplored
- Existing approaches change single triples, but single triples contain little information

```
class ShaleUnit extends GeoUnit (Double temperature, ... )
  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
```

- Domain-specific operations add or change whole subgraphs
- For example, add or remove a whole layer

Testing - Robustness Mask

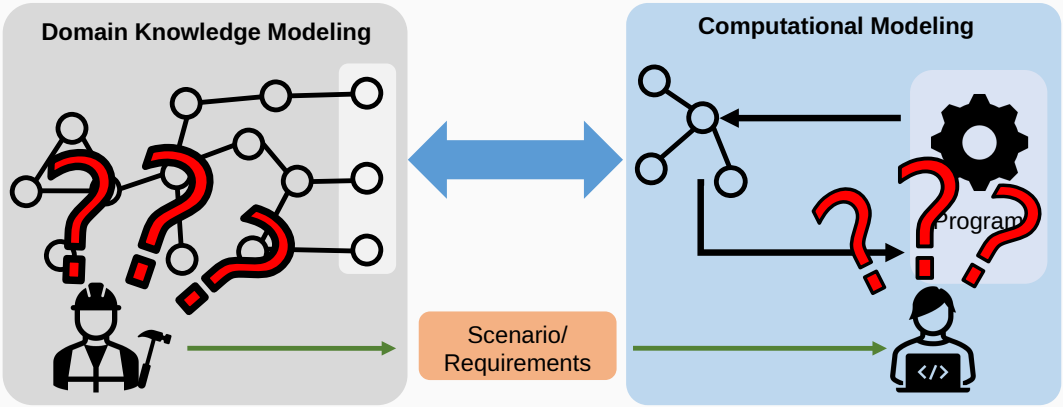
- What does a program exactly need from KG?
- Large parts should be left unchanged (top-level ontology, ...)
- Specify additional checks as SHACL shapes

```
class ShaleUnit extends GeoUnit (Double temperature, ... )
  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
```

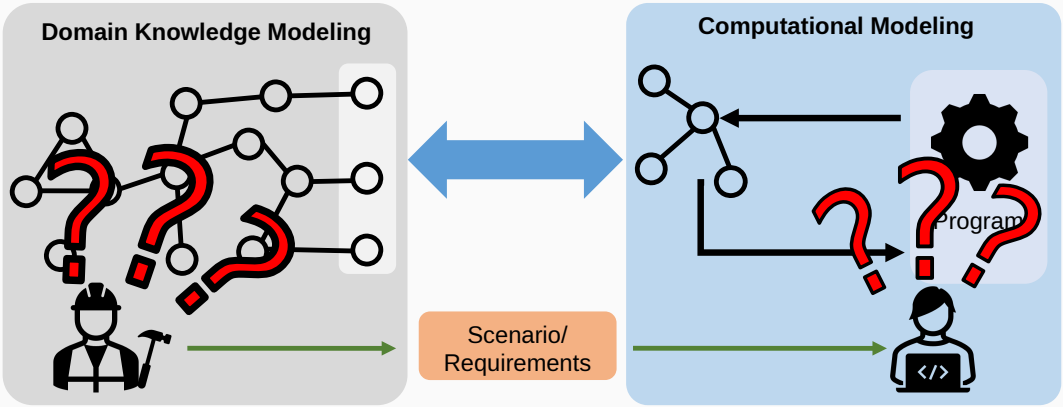
- For example, each new layer must be on top of the old ones
- Kerogen/organic matter cannot occur in all rocks

Correctness – Specification

Functional Correctness



Functional Correctness



Can we use ontologies also for specification of behavior and static verification?

[Kamburjan and Gurov, *A Hoare Logic for Domain Specification*, 2024]

What is a Car?

Suppose you model the assembly process of a car

```
1 procedure addWheels(p) nrWheels := p end
```

What is a Car?

Suppose you model the assembly process of a car

```
1 procedure addWheels(p) nrWheels := p end
```

Programmer

This procedure sets the number of wheels in a car to the value of p.

$$\{\top\}\mathit{addWheels}(p)\{\texttt{nrWheels} \doteq \texttt{p}\}$$

Subject-Matter Expert

I want that in the end of this step, the car classifies as a small car.

$$\{\top\}\mathit{addWheels}(p)\{\texttt{Small}(c)\}$$

What is a Car?

Suppose you model the assembly process of a car

```
1 procedure addWheels(p) nrWheels := p end
```

Programmer

This procedure sets the number of wheels in a car to the value of p.

$\{\top\}\mathit{addWheels}(p)\{\texttt{nrWheels} \doteq \texttt{p}\}$

Subject-Matter Expert

I want that in the end of this step, the car classifies as a small car.

$\{\top\}\mathit{addWheels}(p)\{\texttt{Small}(c)\}$

How to enable both of them to specify their respective intent?

- SME does not know about how the car c is encoded
- Programmer does not know what it means for a car to be small.

Lifted Specification

Ontologies and Description Logics

For domain modeling and specification a rich body of methodologies and tools exist.

$$\texttt{HasFourWheels} \sqsubseteq \texttt{Small} \qquad \exists \texttt{wheels}.\exists \texttt{hasValue}.4 \equiv \texttt{HasFourWheels}$$

Lifted Specification

Ontologies and Description Logics

For domain modeling and specification a rich body of methodologies and tools exist.

$$\texttt{HasFourWheels} \sqsubseteq \texttt{Small} \qquad \exists \texttt{wheels}.\exists \texttt{hasValue}.4 \equiv \texttt{HasFourWheels}$$

$$\left\{ \begin{array}{c} - \\ \mathtt{p} \doteq 4 \end{array} \right\} \texttt{addWheels}(\mathtt{p}) \left\{ \begin{array}{c} \texttt{Small}(c) \\ - \end{array} \right\}$$

- Upper component specifies the state as interpreted in the domain
- Lower component specifies non-lifted state

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{ \quad \right\}$$
$$\texttt{nrWheels} := \texttt{p}$$
$$\left\{ \texttt{Small}(c) \quad \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{ \quad \right\}$$
$$\texttt{nrWheels} := \texttt{p}$$
$$\left\{ \texttt{Small}(c), \texttt{HasFourWheels}(c), \texttt{hasValue}(\texttt{wheelsVar}, 4) \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{ \quad \right\}$$
$$\texttt{nrWheels} := \texttt{p}$$
$$\left\{ \begin{array}{c} \texttt{Small}(c), \texttt{HasFourWheels}(c), \texttt{hasValue}(\texttt{wheelsVar}, 4) \\ \texttt{nrWheels} \doteq 4 \end{array} \right\}$$

Keeping State and Lifted State Connected

Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{ \begin{array}{c} \\ \texttt{p} \doteq 4 \end{array} \right\}$$
$$\texttt{nrWheels} := \texttt{p}$$
$$\left\{ \begin{array}{c} \texttt{Small}(c), \texttt{HasFourWheels}(c), \texttt{hasValue}(\texttt{wheelsVar}, 4) \\ \texttt{nrWheels} \doteq 4 \end{array} \right\}$$

Keeping State and Lifted State Connected

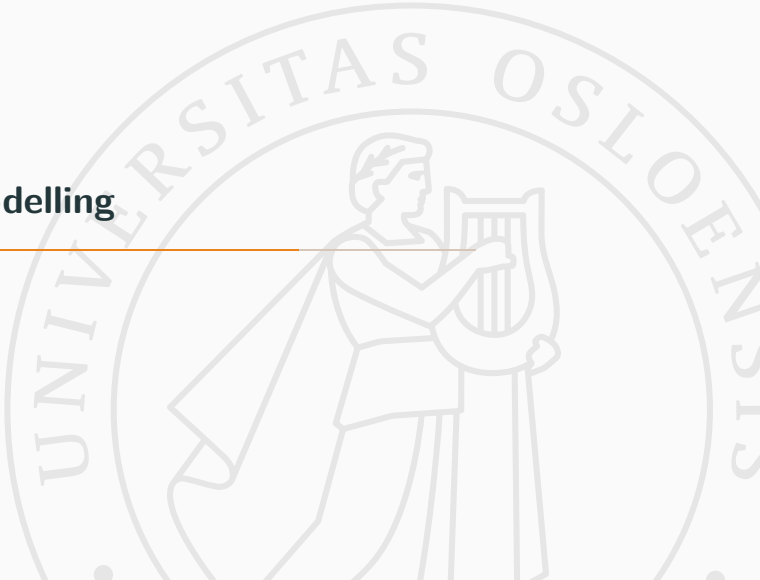
Idea: define a compatible lifting of the specification as well.

Perform following steps for wp reasoning:

1. Infer (abduct/deduct) lifted post-condition
2. Recover state post-condition, substitution
3. Lift pre-condition, deduce domain pre-conditions

$$\texttt{wheels}(c, \texttt{wheelsVar}) \vdash \left\{\begin{array}{c}\texttt{hasValue}(\texttt{pVar}, 4)\\ \texttt{p} \doteq 4\end{array}\right\}$$
$$\texttt{nrWheels} := \texttt{p}$$
$$\left\{\begin{array}{c}\texttt{Small}(c), \texttt{HasFourWheels}(c), \texttt{hasValue}(\texttt{wheelsVar}, 4)\\ \texttt{nrWheels} \doteq 4\end{array}\right\}$$

Asset Modelling



Modeling the Digital Thread

- So far, asset model and digital thread magical: somehow we have all this information in a structure form
- Is this realistic?
- Is this challenge specific to Digital Twins?
- $\Rightarrow$ Asset information modeling is one major field in digitalization right now

Modeling the Digital Thread

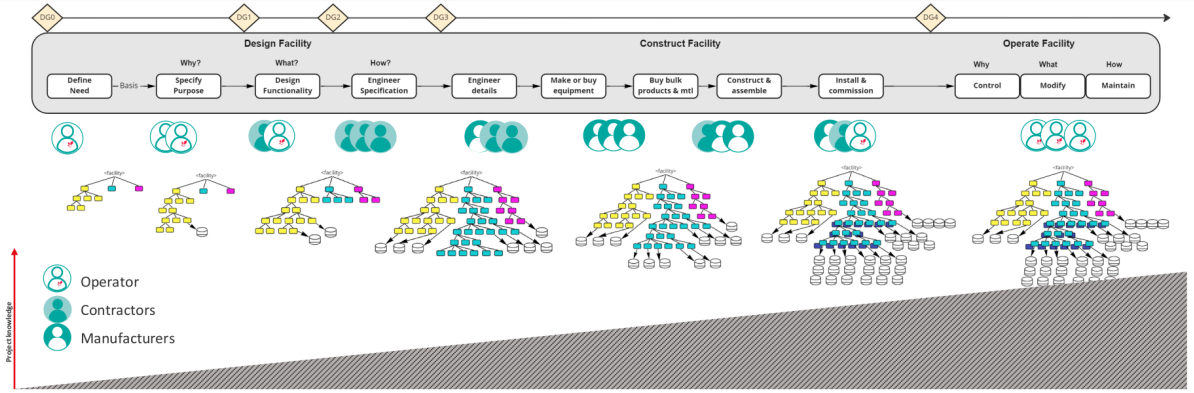
Asset Information Models

Requirements, design and other documents for a built asset, from different perspectives and expressing multiple assets

How to get these asset models?

- Possibility 1: Based on MBSE tooling
  - For example, SysML 2.0 and a knowledge graph export
  - Focuses on development and design process
- Possibility 2: Based on ontological models or reference data languages
  - For example, ISO 23726 (IDO), IEC 63278 (AAS), ISO 15926-14
  - Focuses on data exchange and integration
- Here: *IMF*, SE language developed with roots in ontological models

IMF

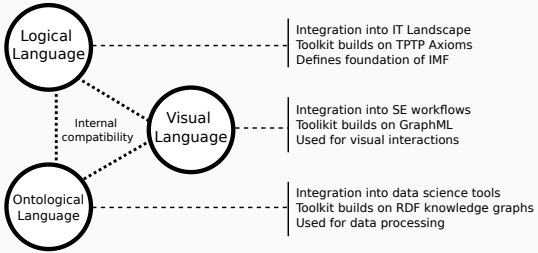

DG0
DG1
DG2
DG3
DG4
Design Facility
Construct Facility
Operate Facility
Define Need
Basis
Why?
Specify Purpose
What?
Design Functionality
How?
Engineer Specification
Engineer details
Make or buy equipment
Buy bulk products & mtl
Construct & assemble
Install & commission
Why
Control
What
Modify
How
Maintain
<facility>
Operator
Contractors
Manufacturers
Project knowledge

IMF - Information modelling framework

Language for identifying and describing objects of industrial assets in different aspects with different syntaxes for different purposes

- Visual block-language for engineering
- Ontological graph-language for data integration
- Formal logic-language for precise constraints

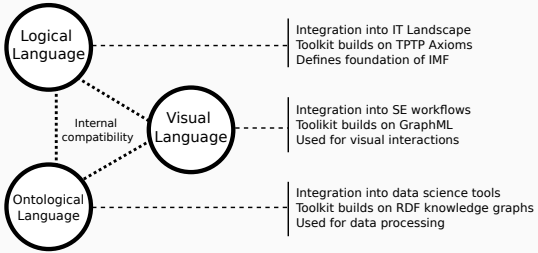
Based on ideas from SE, IEC 81346 aspects, formal logic and knowledge graphs



IMF - Information modelling framework

Language for integrating across ...

- different domain and discipline specific descriptions
- different ontologies and domain standards
- descriptions using different abstraction levels
- the complete asset lifecycle



What IMF is not

- A (full) replacement for established domain standards
  - Keep your existing descriptions, database, diagrams, etc., but ...
  - Use IMF for integration, make these descriptions available between silos
- An upper ontology
  - IMF is a simple/(r) modelling language to identify and relate asset descriptions
  - IMF relies on external standards for semantic definition
- A reference data language (RDL)
  - IMF is an asset description language
  - IMF is designed to allow connections to existing RDLs for generic semantic descriptions
- A language for calculation/simulation/algorithms
  - IMF is a information representation language

IMF Basics

Block 

- Any entity at any abstraction level
- Abstraction mechanism
- Boundary hiding internal details: the insides are not directly accessible to the outside
- Processing black box, transforms media input to output

Examples: *system, function, activity, process, product, area, reservoar, engine, instance, pumping, pump, alarm*

Connector 

- Block connection
- Conditions for connections
- Abstracted block with infinitesimal boundary
- Point of interaction

Examples: *driveshaft, pipe, flange specification, information exchange protocol*

Terminal 

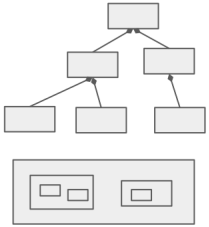
- Block port
- Dependant on 1 block
- Point where medium passes the block boundary
- Conditions on medium
- *How can a block communicate with the outside?*

Examples: *nozzle, cable connection, power in, oil export out, wellstream*

IMF Visual Syntax

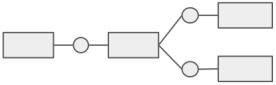
Partonomy

- Part-whole relationships
- *Engine is **part of** Car*
- Tree structure, one parent
- Abstraction by hiding smaller components



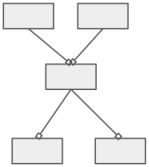
Topology

- Connection/interaction
- *Engine is **connected to** Driveshaft, which is **connected to** Differential*
- Interaction, media flow
- *Driveshaft transfers mechanical energy*



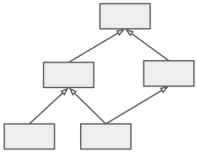
Requirement–Solution

- Fulfillment/"Implementation"
- Requirement is fulfilled by solution; solution is a specialisation of requirement
- *Engine **fulfills** the function of 'converting energy into rotation'. T-Ford Engine is an **implementation** of Engine.*

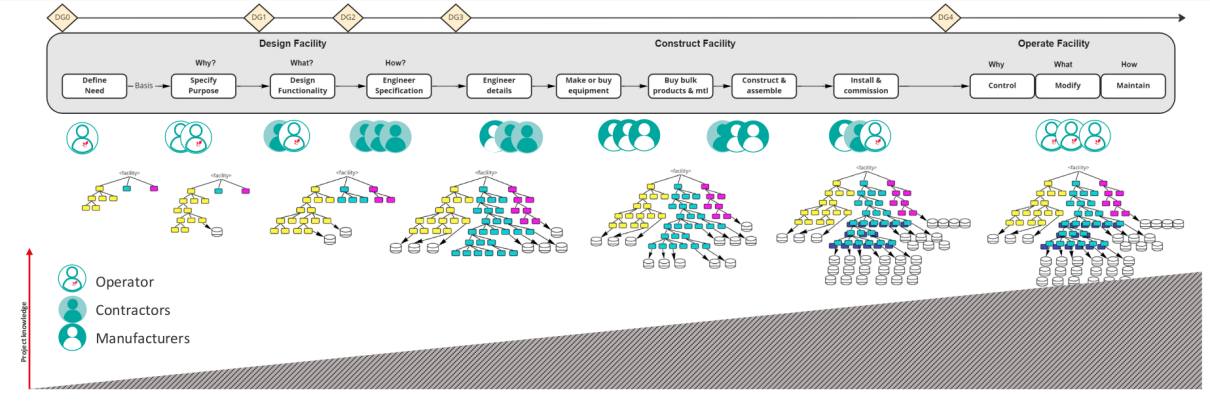


Specialisation

- Specialisation/generalisation of elements
- Inheritance/(reuse) of all(!) features
- *4-stroke engine **is an** Engine*
- Abstraction by hiding specific features

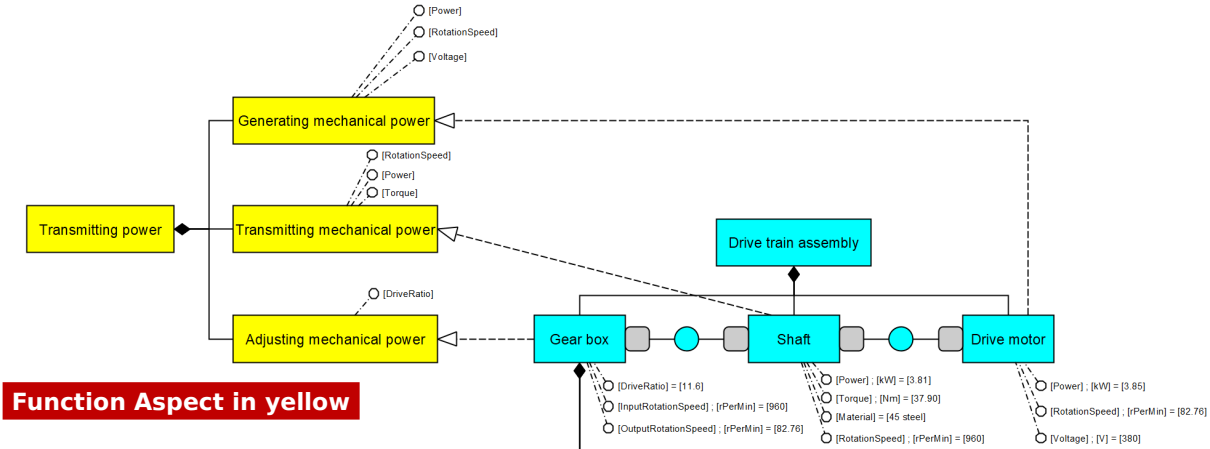


IMF Usage


DG0
DG1
DG2
DG3
DG4
Design Facility
Construct Facility
Operate Facility
Why?
What?
How?
Why
What
How
Define Need
Basis
Specify Purpose
Design Functionality
Engineer Specification
Engineer details
Make or buy equipment
Buy bulk products & mtl
Construct & assemble
Install & commission
Control
Modify
Maintain
Operator
Contractors
Manufacturers
Project knowledge

IMF Aspects

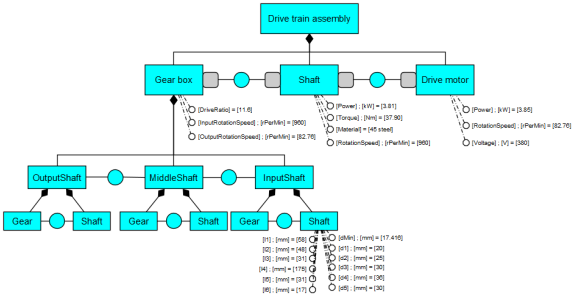
Modality	Interest	Domain	Aspect
Intended	none	Activity	*Function*
		Space	*Location*
		Implementation	*Product*
Actual			Installed
Intended	Project	Activity	Activity need
		Space	Area need
		Implementation	Plant
	Product	Activity	Product function
		Space	Product location
		Implementation	Product implementation



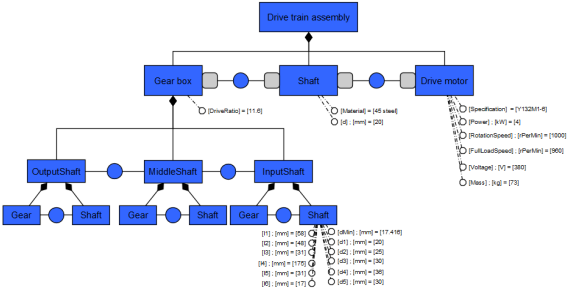
Function Aspect in yellow

Product Aspect in cyan

Product Aspect in cyan



Installed Aspect in dark blue



IMF and Self-adaptive Digital Twins

- IMF has a native export into knowledge graphs
- Can be enriched with ontological terms (e.g., IDO classes for blocks)
- Structures exactly the information we need

Consistency

Installed Aspect models heaters and their simulators in our smart house

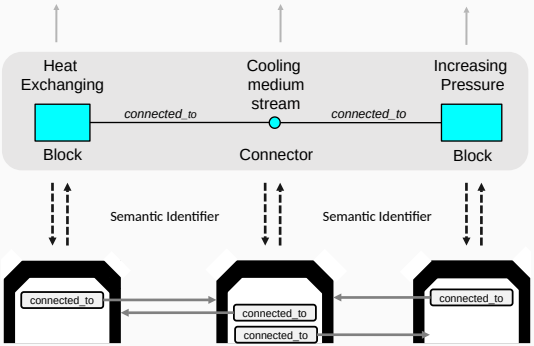
Location Aspect models rooms and their relation to each others

Product Aspect models used requirements for lifecycles

Other Aspects can be added for digital twin specific information

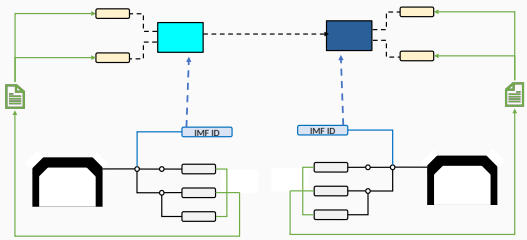
IMF and AAS

- FMI useful for integration, but our approach is more general
- Asset Administration Shells can play the role of simulators, or containers for information
- Instead of managing cross-references, user knowledge graph to model structure
- Ongoing work: comparing AAS across an IMF model, import/export



IMF and AAS

- FMI useful for integration, but our approach is more general
- Asset Administration Shells can play the role of simulators, or containers for information
- Instead of managing cross-references, user knowledge graph to model structure
- Ongoing work: comparing AAS across an IMF model, import/export



IMF - Adaptation

Adaptation In Norway

- Industrial use in Norwegian companies: AIBEL, DNV, AkerBP, AkerSolutions, Equinor
- Quasi-Standard as DNV recommended practice (DNV-RP-0670, to be published)
- Motivation: standardized asset modeling outside pure engineering with ability to relate to external ontological standards

- Used as exchange formalism for dataspaces in EU projects SM4RTENANCE and Tec4MaaSEs
- Completely open: toolkit and documentation available under `www.imfid.org`
- v1.0 release in the next months

Conclusion

Summary

- **Part I** Semantic Reflection

  How to connect software with the digital thread?

- **Part II** Self-Adaptation in Digital Twins

  How to use asset models for structural self-adaptation?

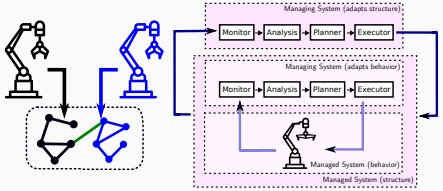
- **Part III** Correctness

  How to ensure type safety, functional correctness, and test digital twins?

- **Part IV** Asset Information Modeling

  How to develop asset models?

Summary

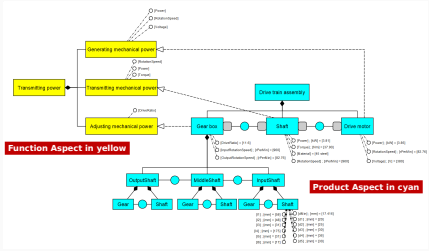


Semantic Lifting and Digital Twins

- Twin structure as knowledge graph connected with digital thread. Consistency as graph queries.
- Structural self-adaptation through two-layer MAPE-K loop

Standard-Driven Asset Information Modeling

Describing structure, requirements and simulators as aspect-oriented digital thread



- `www.smolang.org`
- `www.imfid.org`