Perspectives on Consistency for System and Software Contracts

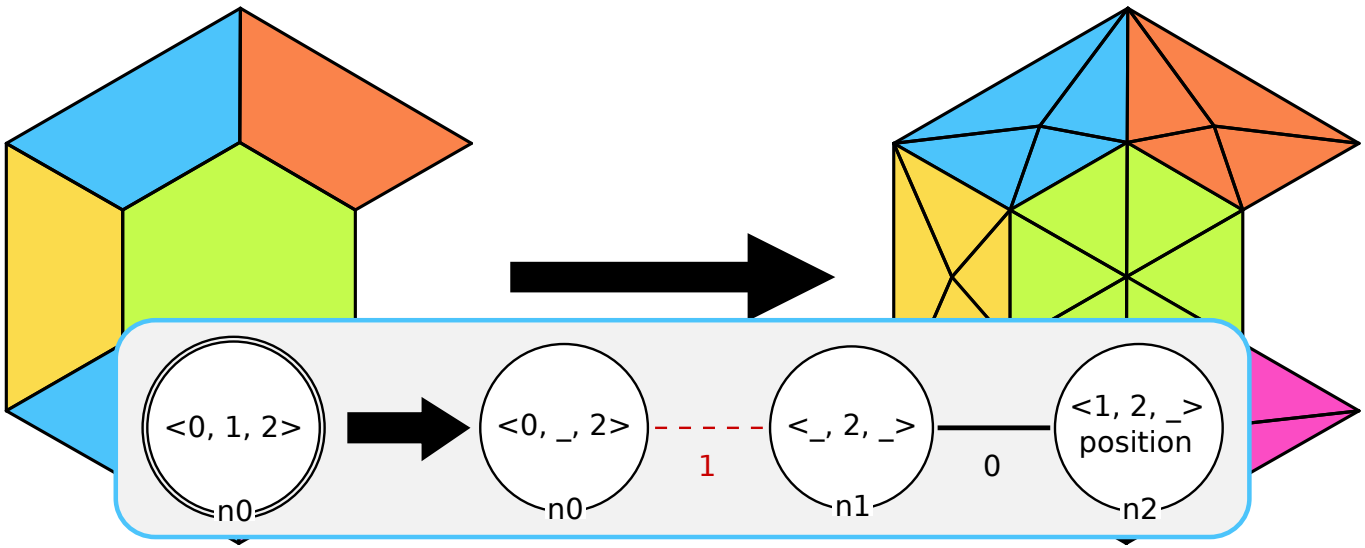
Romain Pascual

January 13, 2026

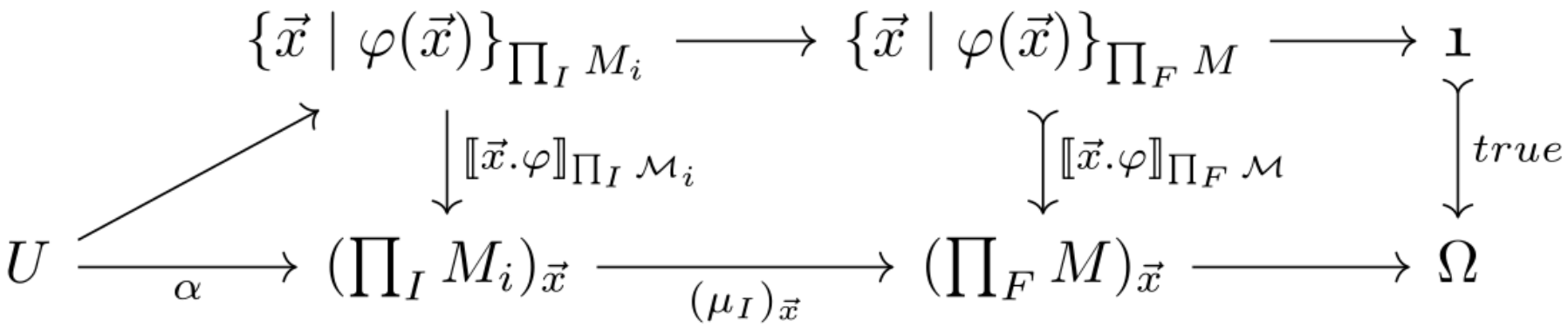
Dagstuhl Seminar 26031

CentraleSupélec | université PARIS-SACLAY

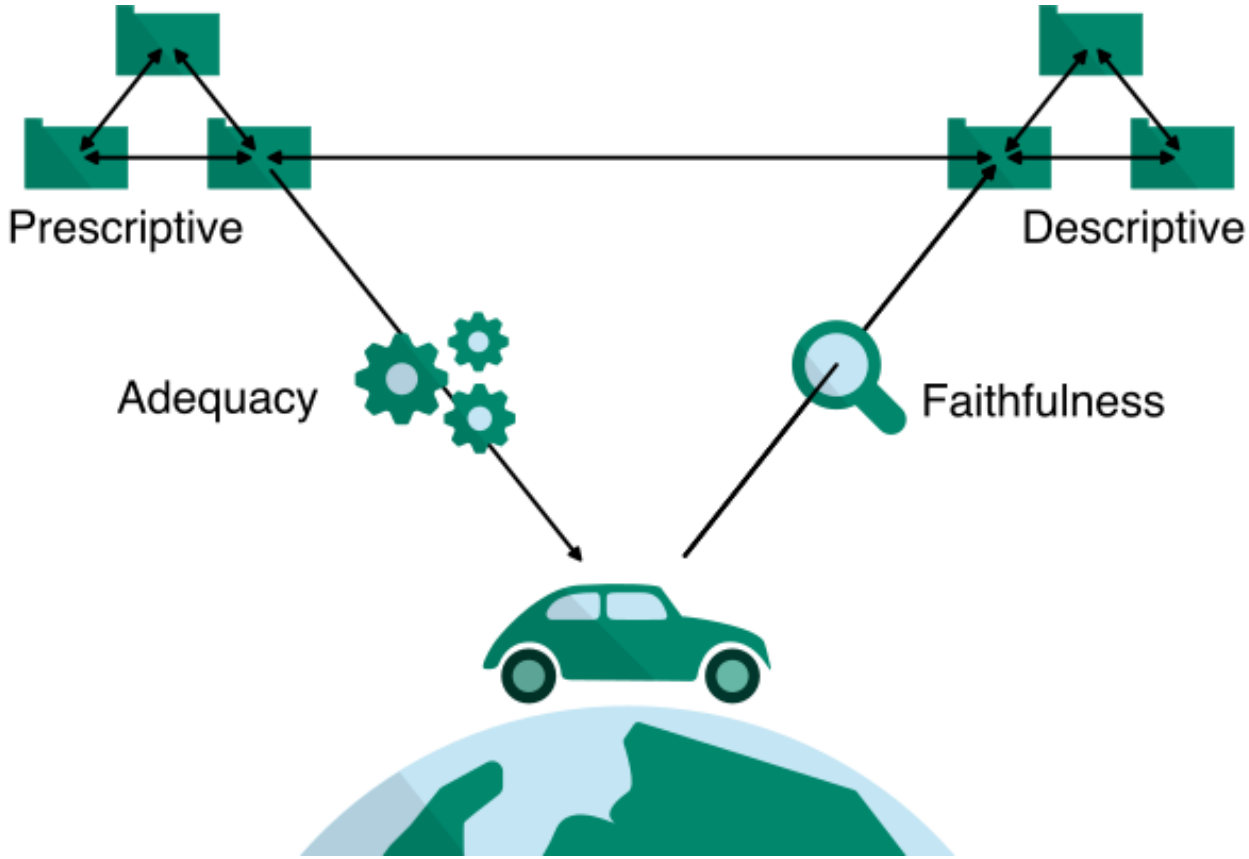
About Me



About Me


$\{\vec{x} \mid \varphi(\vec{x})\}_{\prod_I M_i}$
$\{\vec{x} \mid \varphi(\vec{x})\}_{\prod_F M}$
$\mathbf{1}$
$[\![\vec{x}.\varphi]\!]_{\prod_I \mathcal{M}_i}$
$[\![\vec{x}.\varphi]\!]_{\prod_F \mathcal{M}}$
$true$
U
α
$(\prod_I M_i)_{\vec{x}}$
$(\mu_I)_{\vec{x}}$
$(\prod_F M)_{\vec{x}}$
Ω

About Me



My background alternates between rewriting, semantics, and model consistency: these are my entry points today.

How can system and software contracts be harmonized?

Two Ways to Harmonize

Abstraction

- Unify notions at a higher level
- Single abstract framework
- Risk: losing domain meaning

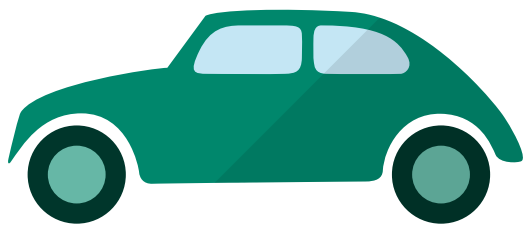
Bridges

- Keep viewpoints distinct
- Identify shared notions
- Enable integration

Consistency appears both in abstraction and in bridges.

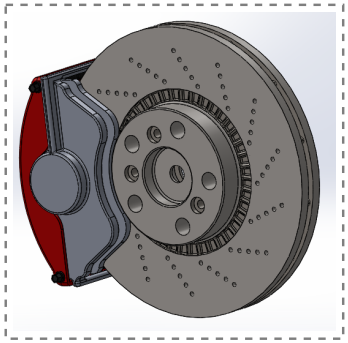
Designing Systems with Models

The Example of the Virtual Single Underlying Model methodology

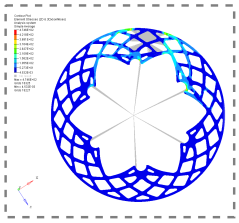


Designing Systems with Models

The Example of the Virtual Single Underlying Model methodology

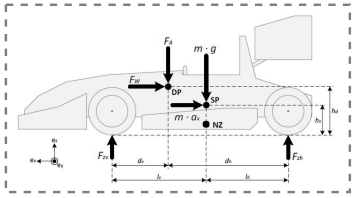


3D Brake Assembly View

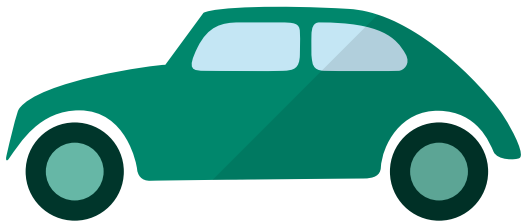


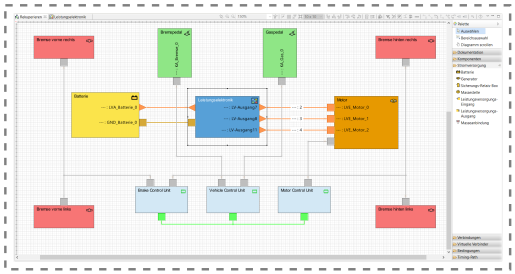
Tribological Simulation View

X-in-the-Loop Test Deployment View

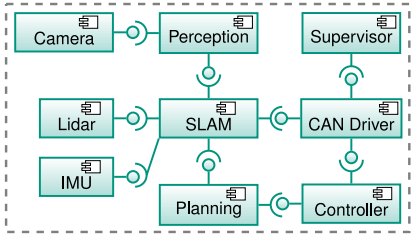


Simplified Vehicle Model View





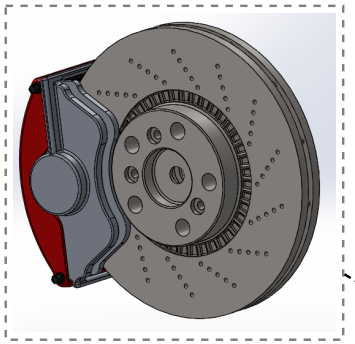
E/E Topology View (PREEvision)



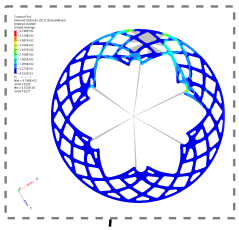
Autonomous Driving Components View

Designing Systems with Models

The Example of the Virtual Single Underlying Model methodology

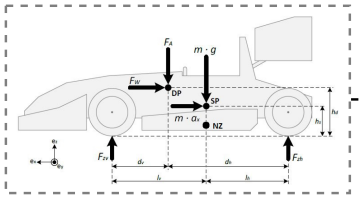


3D Brake Assembly View

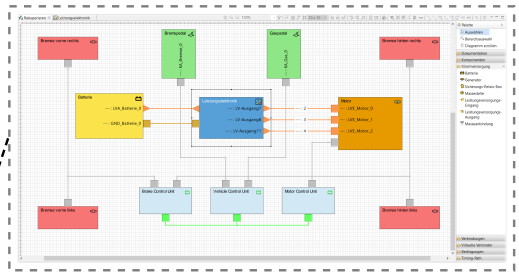


Tribological Simulation View

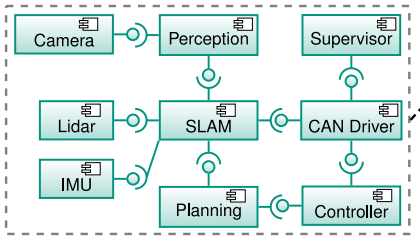
X-in-the-Loop Test Deployment View



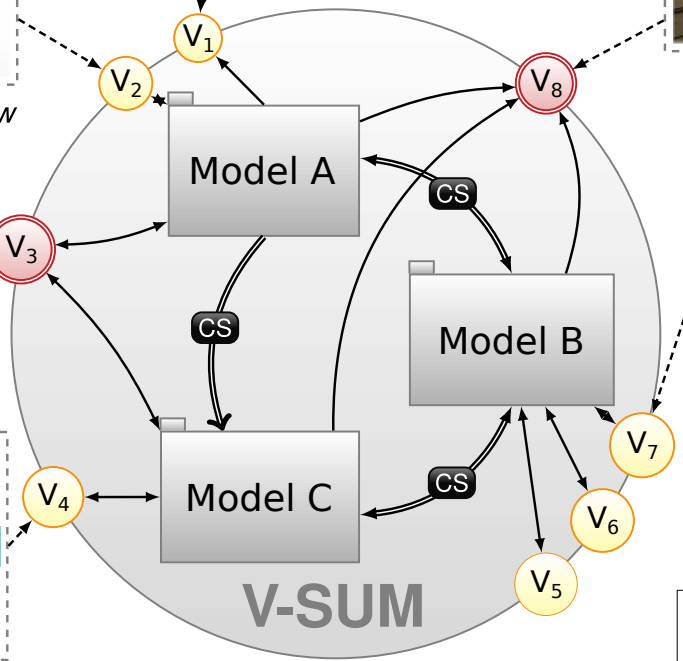
Simplified Vehicle Model View

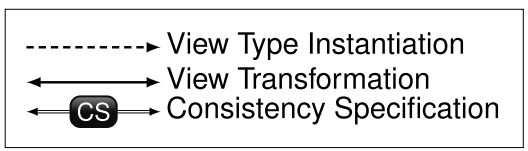


E/E Topology View (PREEvision)



Autonomous Driving Components View





Models, Semantics, Consistency

Model: an abstract representation of an original for a given purpose

Models, Semantics, Consistency

Model: an abstract representation of an original for a given purpose

Models

- UML diagrams
- Automata
- Differential equations
- Source code
- Sketches on a napkin

Models, Semantics, Consistency

Model: an abstract representation of an original for a given purpose

Models

- UML diagrams
- Automata
- Differential equations
- Source code
- Sketches on a napkin

Semantics

(ex. for Java programs)

- Traces
- Pre/post
- Test outcomes
- Quantitative properties

Models, Semantics, Consistency

Model: an abstract representation of an original for a given purpose

Models

- UML diagrams
- Automata
- Differential equations
- Source code
- Sketches on a napkin

Semantics

(ex. for Java programs)

- Traces
- Pre/post
- Test outcomes
- Quantitative properties

Consistency

- "System can be built"
- Preservation by transformations
- Inability to derive falsity (proof-theoretic)
- Existence of models (model-theoretic)

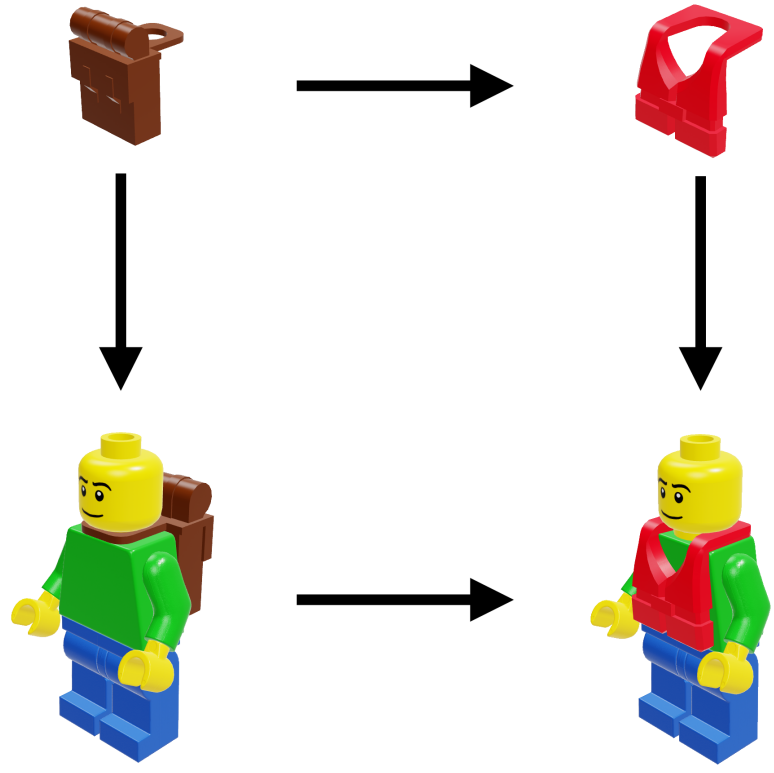
Abstraction

Models	**Semantics**	**Consistency**
$m \in M$	$[\![\cdot]\!] : M \to S$	$\sim \subseteq M \times M$

Abstraction lets us reason about models, semantics, and consistency **without committing** to a specific formalism.

Operations on Models

- How to modify them?



Contracts for Operations on Models

Model repair

$$\{\top\} \qquad (A', B') := \mathsf{repair}(A, B) \quad \{A' \sim B\}$$

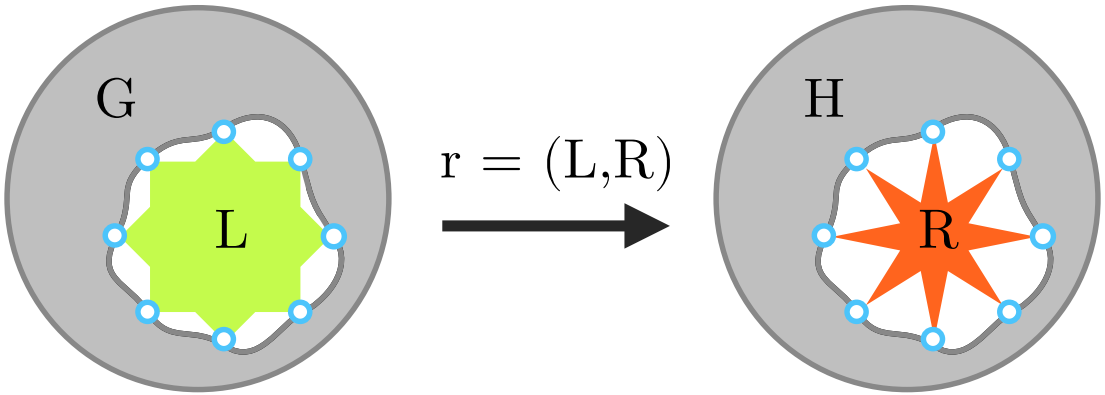
Model slicing

$$\{A \sim B\} \qquad A_s := \mathsf{slice}_B(A) \qquad \{A_s \sim B\}$$

$$\{A \not\sim B\} \qquad A_s := \mathsf{slice}_B(A) \qquad \{A_s \not\sim B\}$$

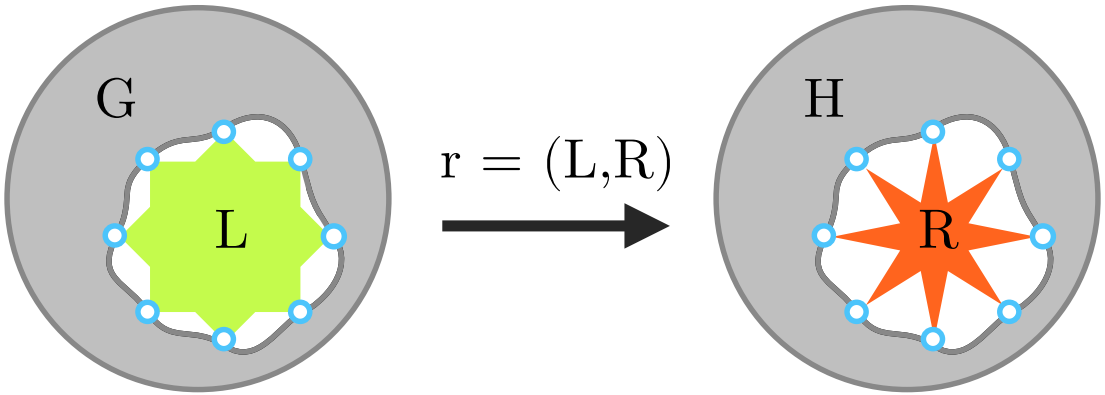
Operations as Graph Rewriting

Models as graphs $\Rightarrow$ model operations as graph transformations.



Operations as Graph Rewriting

Models as graphs $\Rightarrow$ model operations as graph transformations.



Pre	**Operation**	**Post**
$L \subseteq G$	$L \to R$	$R \subseteq H$

Structural conditions on graphs, not temporal conditions on executions.

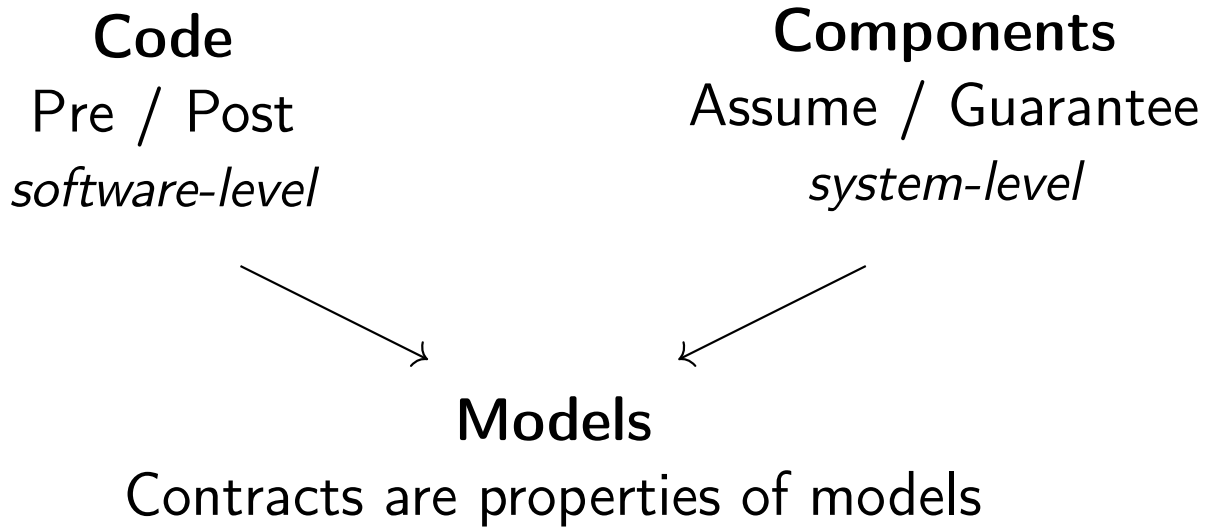
1. Graph rewriting already comes with a **contract-like structure**.
2. Contracts need not be about behavioral properties.

Contracts for Code and Components

Code
Pre / Post
software-level

Components
Assume / Guarantee
system-level

Contracts for Code and Components



Can harmonization happen at the level of the underlying models?

Contracts as Models

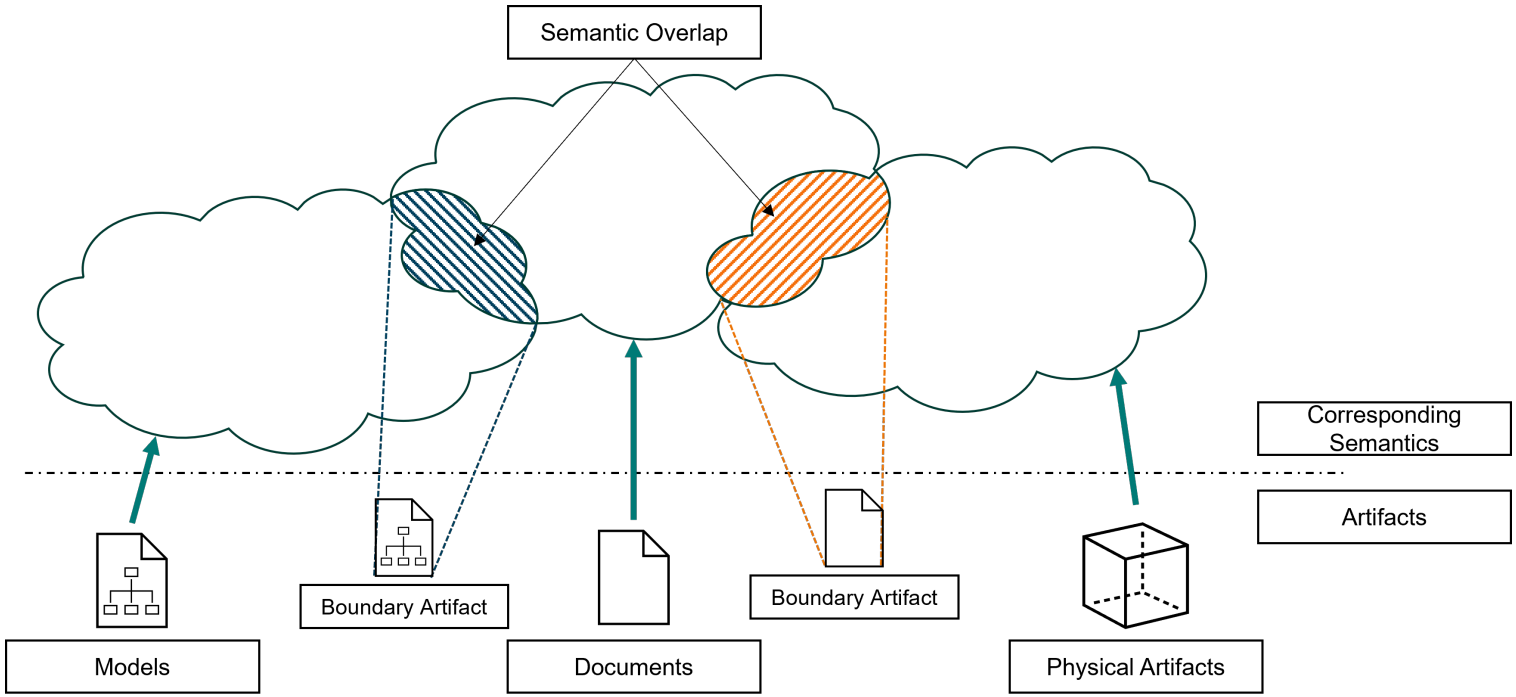
Contracts can be seen as **models** capturing **assumptions** and **guarantees**:

- **Compatibility** of contracts: **joint consistency**.
- **Refinement** of contracts: **preservation of consistency**.

Consistency problem appears whether we call the artifacts **models**, **views**, or **contracts**.

Where Harmonization Becomes Difficult

How to talk to each other?



Different communities talk about the same things with different words.

Harmonization is less about tools, more about agreeing on **properties**.

Open Questions

- Could **consistency** be the right unifying notion to harmonize system and software contracts?
- When are contract inconsistencies design errors – and when are they **useful signals**?
- What should be made **explicit**, and what should remain **implicit**?