Semantic Consistency in Model-Driven Development

SETSS 2026

Tianhai Liu[1] Shmuel Tyszberowicz[2] Bernhard Beckert[1]

[1]Karlsruhe Institute of Technology (KIT) | [2]Afeka Academic College of Engineering (AFEKA)

DFG – SFB 1608 – 501798263, CONVIDE | May 2026

What is consistency of models?

Pragmatic intuition

Models are consistent if they can be jointly realised.

What is consistency of models?

Pragmatic intuition

Models are consistent if they can be jointly realised.

Relational view

Let $\mathrm{MM}_1, \ldots, \mathrm{MM}_n$ be meta-models, interpreted as sets of valid models. A consistency relation is

$$\mathrm{CR} \subseteq \mathrm{MM}_1 \times \cdots \times \mathrm{MM}_n.$$

Models $m_1 \in \mathrm{MM}_1, \ldots, m_n \in \mathrm{MM}_n$ are consistent iff

$$(m_1, \ldots, m_n) \in \mathrm{CR}.$$

Consistency Specification

From relation to specification

A consistency relation defines which model tuples are consistent:

$$CR \subseteq MM_1 \times \cdots \times MM_n$$

A *consistency specification* is an artefact that induces such a relation.

How to specify

- constraints or predicates
- model transformations
- consistency checkers
- formal semantics
- domain-specific rules

Consistency becomes explicit by specifications that tools and engineers can use.

Consistency Preservation Rules

Before a change

The models are consistent:

$$(m_1, \ldots, m_n) \in \text{CR}$$

Then a developer changes one or more models:

$$\delta_1, \ldots, \delta_n$$

After preservation

A preservation rule derives adapted changes:

$$\delta'_1, \ldots, \delta'_n$$

such that consistency is restored:

$$(\delta'_1(m_1), \ldots, \delta'_n(m_n)) \in \text{CR}$$

Models in CPS

- Heterogeneous multidisciplinary models and analyses.
- Model changes cause inconsistency silently.

Models in CPS

- Heterogeneous multidisciplinary models and analyses.
- Model changes cause inconsistency silently.

Syntactic Consistency

- Names match
- References resolve
- Types are compatible
- Constraints over model elements hold

Models in CPS

- Heterogeneous multidisciplinary models and analyses.
- Model changes cause inconsistency silently.

Syntactic Consistency

- Names match
- References resolve
- Types are compatible
- Constraints over model elements hold

Semantic Consistency

- Meanings agree
- Behaviours are compatible
- Analyses use coherent assumptions
- **Overlapping** system properties are not contradicted

From SUM to *V-SUM*

SUM - Single Underlying Models

- Views over one single monolithic model
- Consistency by construction

From SUM to *V-SUM*

SUM - Single Underlying Models

- Views over one single monolithic model
- Consistency by construction

V-SUM - Virtual Single Underlying Models

[Ralf Reussner+'21]

- Multiple heterogeneous models
- Diverse specialised tools
- Cross-view dependencies

Analyses as First-Class Citizens

Heterogeneous models and analyses

- **Mechanical domain:**
  - Multi-body dynamics simulation → Mechanical models (mass, force)
  - Fault Tree Analysis (FTA) → System failure models
- **Electrical domain:**
  - Circuit simulation → Circuit models (voltage, current)
  - Failure Mode and Effects Analysis (FMEA) → Component failure models
- **Software domain:**
  - Verification → Behavioural models (state machines, specs)
  - Static code analysis → Source code
  - Code reviews and compliance checks → code, guidelines

Analyses as First-Class Citizens

Heterogeneous models and analyses

- **Mechanical domain:**
  - Multi-body dynamics simulation → Mechanical models (mass, force)
  - Fault Tree Analysis (FTA) → System failure models
- **Electrical domain:**
  - Circuit simulation → Circuit models (voltage, current)
  - Failure Mode and Effects Analysis (FMEA) → Component failure models
- **Software domain:**
  - Verification → Behavioural models (state machines, specs)
  - Static code analysis → Source code
  - Code reviews and compliance checks → code, guidelines

After model changes

- Which previous analysis results remain valid?
- Which analyses must be rerun?

Analyses as First-Class Citizens

Heterogeneous models and analyses

- **Mechanical domain:**
  - Multi-body dynamics simulation → Mechanical models (mass, force)
  - Fault Tree Analysis (FTA) → System failure models
- **Electrical domain:**
  - Circuit simulation → Circuit models (voltage, current)
  - Failure Mode and Effects Analysis (FMEA) → Component failure models
- **Software domain:**
  - Verification → Behavioural models (state machines, specs)
  - Static code analysis → Source code
  - Code reviews and compliance checks → code, guidelines

After model changes

- Which previous analysis results remain valid?
- Which analyses must be rerun?

Current practice is often conservative, leading to excessive reruns and delayed certification and updates.

Gap in current *V-SUM* theory

V-SUM integrates heterogeneous models through explicit mappings and consistency relations for multiple metamodels.

What existing *V-SUM* does well

- Models as first-class artefacts
- Consistency relations across models
- Propagation of model (syntactical) changes via consistency preservation rules

Gap in current *V-SUM* theory

V-SUM integrates heterogeneous models through explicit mappings and consistency relations for multiple metamodels.

What existing *V-SUM* does well

- Models as first-class artefacts
- Consistency relations across models
- Propagation of model (syntactical) changes via consistency preservation rules

But

- Analyses are mostly external to V-SUM
- Dependencies between analyses are implicit
- No analysis-aware consistency management
- No formal basis for selective re-execution

Gap in current *V-SUM* theory

V-SUM integrates heterogeneous models through explicit mappings and consistency relations for multiple metamodels.

What existing *V-SUM* does well

- Models as first-class artefacts
- Consistency relations across models
- Propagation of model (syntactical) changes via consistency preservation rules

But

- Analyses are mostly external to V-SUM
- Dependencies between analyses are implicit
- No analysis-aware consistency management
- No formal basis for selective re-execution

Our goal is to make analyses first-class citizens alongside models.

Formal model of an analysis

$$f : I \otimes C \otimes P \rightarrow \mathit{Validity} \otimes \mathit{Evidence}$$

Input

- I: input models subject to analysis
- C: preconditions, e.g., Java installed for JUnit tests
- P: properties for I, expressed with observables

Output

- $\mathit{Validity} \in \{\text{true}, \text{false}\} \Leftrightarrow i \vDash p$
- Evidence: proofs, traces, CEs, reports, ...

Observables: measurable or derivable properties of models (e.g. physical quantities, constraints).

Model relations between input artefacts and reusable analysis results.

Compositionality

Compositionality

Local analyses achieve local goals:

$$f_{g_1} \vDash g_1, \ldots, f_{g_n} \vDash g_n$$

Together they entail a global goal:

$$\bigwedge_i (f_{g_i} \vDash g_i) \Longrightarrow f_G \vDash G$$

Essential for safety assurance cases (e.g., ISO 26262): certification goals are built from local analyses.

Model consistency defined via analyses

General view

A tuple of heterogeneous models

$$m = \langle m_1, \ldots, m_n \rangle$$

is consistent w.r.t. a global goal G ($m \in CR_G$) if its set of analyses

$$F = \{f_{g_1}, \ldots, f_{g_n}\}$$

achieves local goals g_i and

$$\bigwedge_{i=1}^{n} g_i \Longrightarrow G$$

Observable-based view

- Analyses evaluate properties over observables of models
- Consistency requires compatibility of observables across models

$$CR_G = \left\{ \langle m_1, \ldots, m_n \rangle \;\middle|\; (Obs(m_1), \ldots, Obs(m_n)) \in R_G \right\}$$

R_G: cross-model compatibility relation of observables for goal G

Analysis Dependency Graph (ADG)

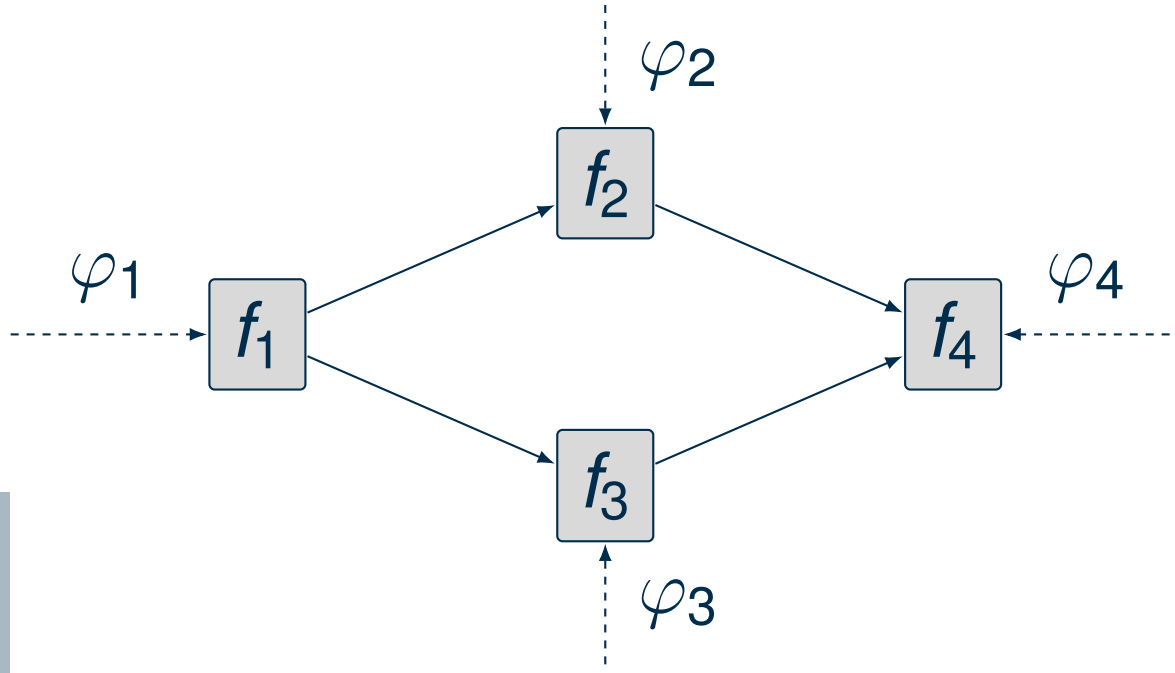
An ADG is a directed graph

$$A = (V, E, \Phi)$$

where:

- nodes = instantiated analyses
- edges = activation and dependency relations
- Φ = trigger functions inspect the model delta on observables. $\varphi(\Delta_{\text{weight}}) = |\Delta_{\text{weight}}| \geq 10$

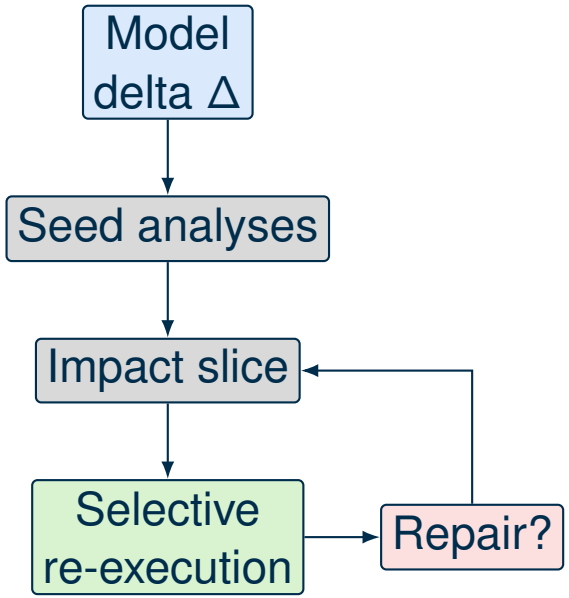
A change does *not* necessarily require a full rerun. Analysis slice detected using triggers and use-define relations over observables.



Incremental recertification by analysis slicing

1. **Find initially affected analyses by triggers**
2. **Compute the impact analysis slice (based on use-define relations over observables)**
3. **Rerun only the impact slice topologically**
4. **If repair introduces new deltas, slice again**

Only the semantically affected subset (i.e., core analyses) is re-executed.



Core analysis theorem

$$(F \models G) \iff (F_{\text{core}} \models G)$$

Why safe?

Analyses outside the slice:

- are unaffected by the change
- would reproduce the same result

Why complete?

All analyses that could influence G:

- are reached via dependencies
- are included in the slice

Integration into *V-SUM*

$$V\text{-}SUM^{+} = \langle M,\ F,\ CR,\ CR^{+} \rangle$$

Classical *V-SUM*

- ***M*: domain models**
- ***CR*: domain-to-domain consistency relations**

Our extension

- ***F*: analysis models**
- ***CR*$^{+}$: domain-to-analysis relations via triggers**

Result: consistency management becomes analysis-aware, not only model-aware.

Preliminary Evaluation

Model Change (Δ)	Re-executed	Saved
Software-development		
Comment-only change in code	f_{rel}	4
Decrease of coverage threshold ($\theta_{cov} \downarrow$)	f_{rel}	4
Method added	f_{stat}, f_{cov}, f_{rel}	2
Method removed	f_{stat}, f_{test}, f_{rel}	2
Test added	f_{test}, f_{rel}	3
Test removed	f_{cov}, f_{rel}	3
Safety-critical code changed	f_{stat}, f_{rev}, f_{rel}	2
Guideline rule added	f_{rev}, f_{rel}	3
Increase of coverage threshold ($\theta_{cov} \uparrow$)	f_{cov}, f_{rel}	3
Sum		**26/45**
Average (9 changes)		**2.9/change (58%)**
ISO 26262 safety-assurance		
Hazard or ASIL added	f_{hara}, f_{req}, f_{ver}	0
New or modified safety requirement	f_{req}, f_{ver}	1
Verification artefacts changed	f_{ver}	2
Sum		**3/9**
Average (3 changes)		**1.0/change (33%)**

Consistency between Requirements

Requirements come first

- Early system intent
- Shared stakeholder view
- Basis for models and code
- Mostly natural language

Consistency between Requirements

Requirements come first

- Early system intent
- Shared stakeholder view
- Basis for models and code
- Mostly natural language

But, CPS requirements are rarely isolated

- Documents overlap
- Terms vary
- Ranges may conflict
- Errors propagate downstream

Consistency between Requirements

Requirements come first

- Early system intent
- Shared stakeholder view
- Basis for models and code
- Mostly natural language

But, CPS requirements are rarely isolated

- Documents overlap
- Terms vary
- Ranges may conflict
- Errors propagate downstream

Late-discovered requirement inconsistencies propagate into models, code, and tests.

Final integration stage ⇒ expensive repair.

Example: Braking-system requirements

```
The braking system consists of a mechanical pedal/booster
assembly, an electronic sensing unit, and a software
controller. The driver's pedal effort is converted
mechanically into brake actuation force. The electronic
unit measures the resulting pedal force and produces a
digitised signal. The software controller consumes the
sensor signal and computes a brake torque command, which
is then provided to the actuator.
=== Mechanical Requirements Specification ===
MRS-001: The pedal effort produced by the driver shall be
within the range [0, 300] N.
=== Electronic Requirements Specification ===
ERS-001: The measured pedal force reported by the sensor
shall lie within the range [400, 500] N.
=== Software Requirements Specification ===
SRS-001: The brake torque command shall be valid whenever
the reported pedal force is at least 50 N.
```

Mechanical engineers use **pedal effort**
Electronics engineers use **measured pedal force**
Software engineers use **reported pedal force**

Are these terms synonyms?

Example: Braking-system requirements

```
The braking system consists of a mechanical pedal/booster
assembly, an electronic sensing unit, and a software
controller. The driver's pedal effort is converted
mechanically into brake actuation force. The electronic
unit measures the resulting pedal force and produces a
digitised signal. The software controller consumes the
sensor signal and computes a brake torque command, which
is then provided to the actuator.
=== Mechanical Requirements Specification ===
MRS-001: The pedal effort produced by the driver shall be
within the range [0, 300] N.
=== Electronic Requirements Specification ===
ERS-001: The measured pedal force reported by the sensor
shall lie within the range [400, 500] N.
=== Software Requirements Specification ===
SRS-001: The brake torque command shall be valid whenever
the reported pedal force is at least 50 N.
```

Mechanical engineers use **pedal effort**
Electronics engineers use **measured pedal force**
Software engineers use **reported pedal force**

Are these terms synonyms? In the context:

$$[0, 300] \cap [400, 500] = \emptyset$$

Requirements are inconsistent

Recall Observables

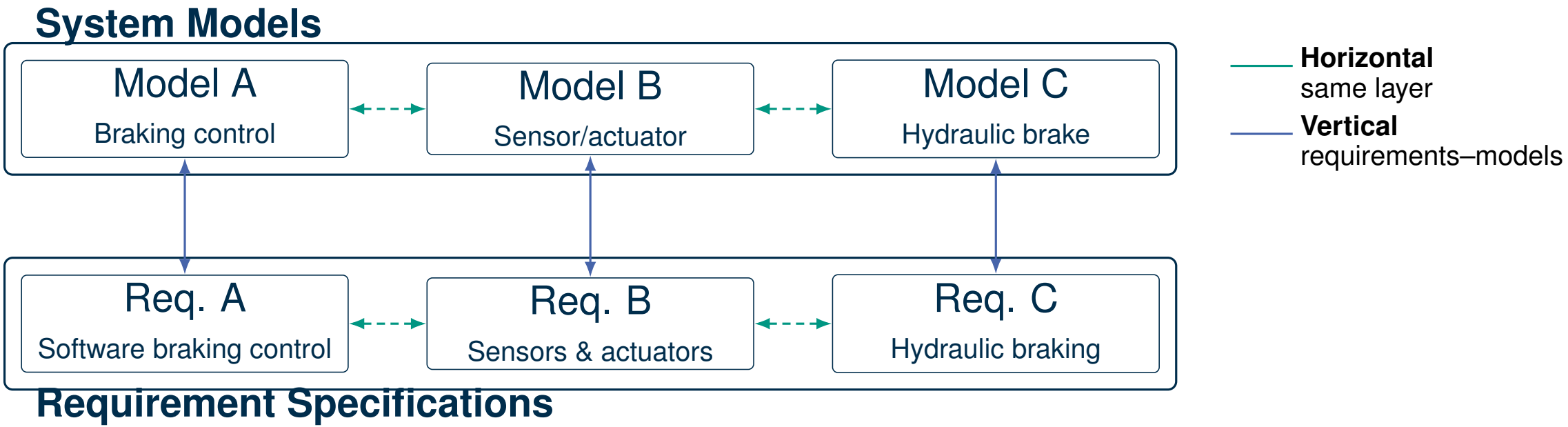
$$o = \langle n, T, C \rangle$$

- n: observable name
- T: observable type: numerical or categorical
- C: constraints over o (e.g., $[0, 300] \cup [400, 500]$)

$$\mathcal{O} = \{ o_1, \ldots, o_k \}$$

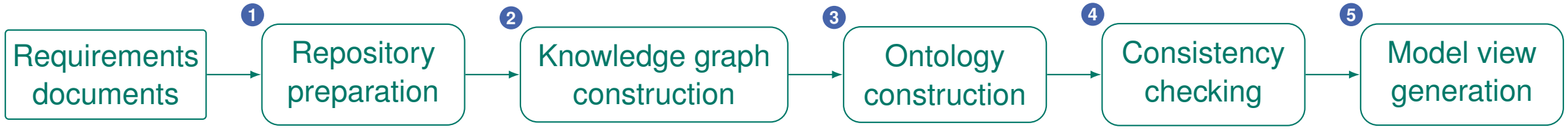
- Shared by requirements and models
- Common observable space between requirements and models
- Basis for solver-based reasoning

Horizontal and Vertical Consistency



All consistency relations are checked through shared observables.

RAG4C: A Semantics-Grounded Pipeline



LLMs extract; ontologies align; solvers decide.

Stage 1: Repository Preparation

What we remove

- Headers and footers
- Page numbers
- Administrative metadata
- Figures and diagrams

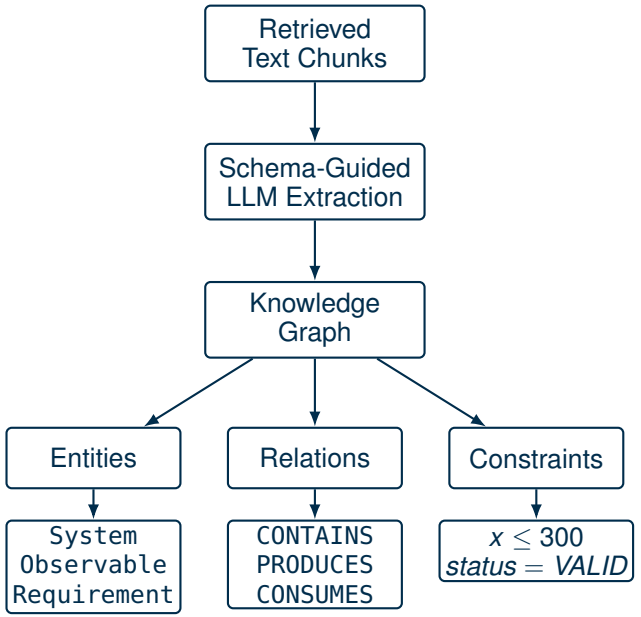
What we keep

- System descriptions
- Requirement statements
- Observable-bearing text
- Source document identifiers

Outcome

A clean, traceable corpus for retrieval-augmented observable extraction.

Stage 2: Knowledge Graph Construction



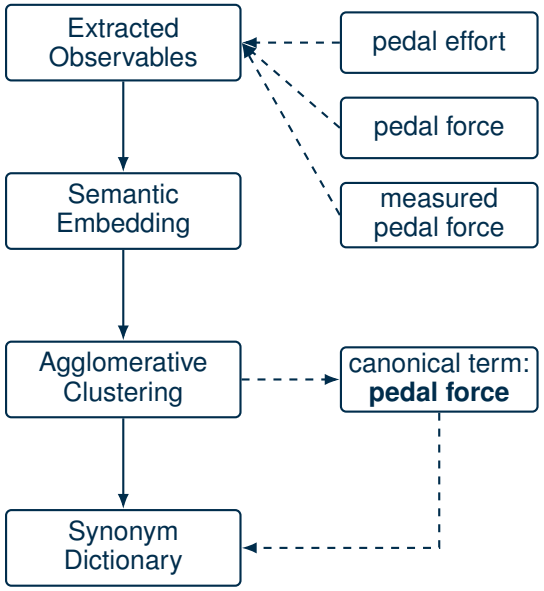
What happens?

- Requirements are processed as retrieval chunks
- LLM extracts only allowed entity and relation types
- Constraints are normalised into a machine-readable form

Outcome

A traceable knowledge graph that links requirement text to observables, systems, relations, and constraints.

Stage 3: Ontology Construction



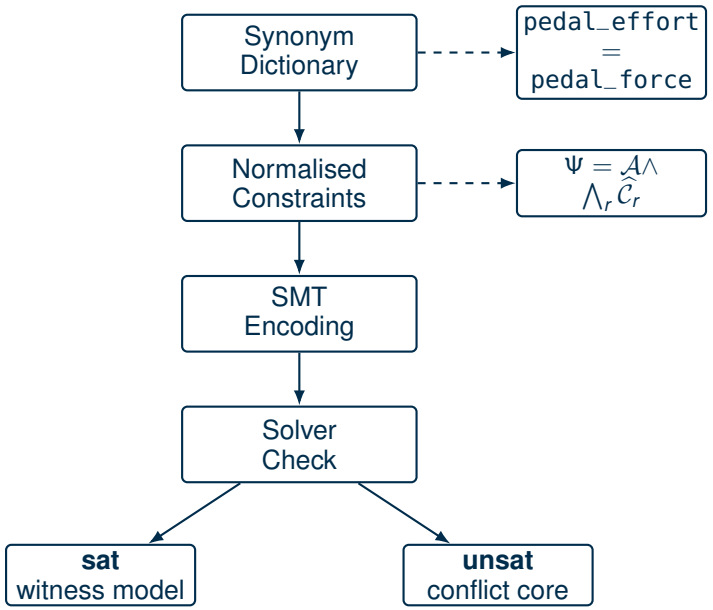
What happens?

- Extracted observables are embedded with their context
- Semantically close terms are clustered
- Each cluster receives one canonical ontology term
- Raw names remain traceable through the synonym dictionary

Outcome

A semantic vocabulary with unified terms.

Stage 4: Consistency Checking



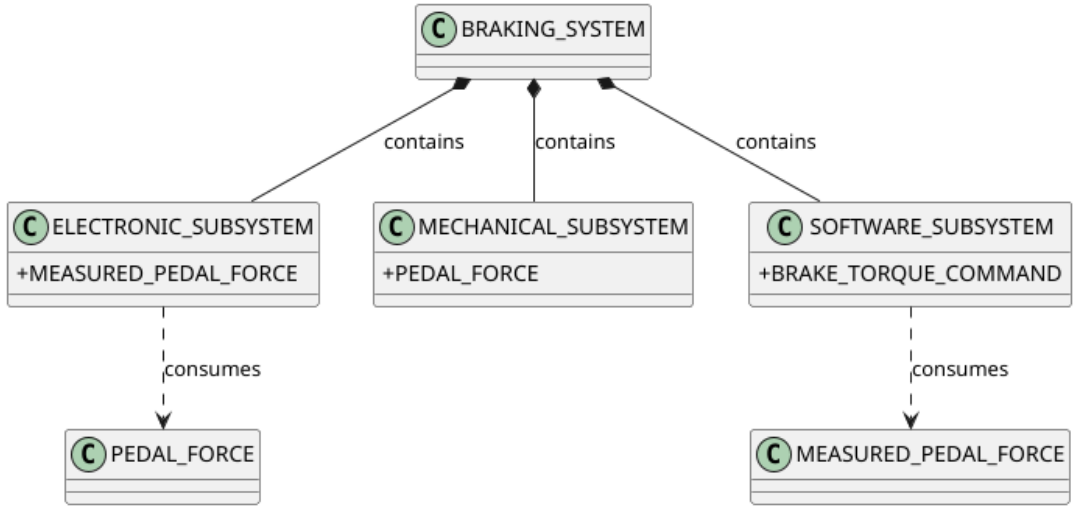
What happens?

- Aliases are encoded as equalities
- All constraints are rewritten using canonical observable names
- The conjunction is checked by an SMT solver
- Results are mapped back to source requirements

Outcome

Consistency becomes a satisfiability question, with evidence for both consistency and inconsistency.

Stage 5: Model View Generation



What happens?

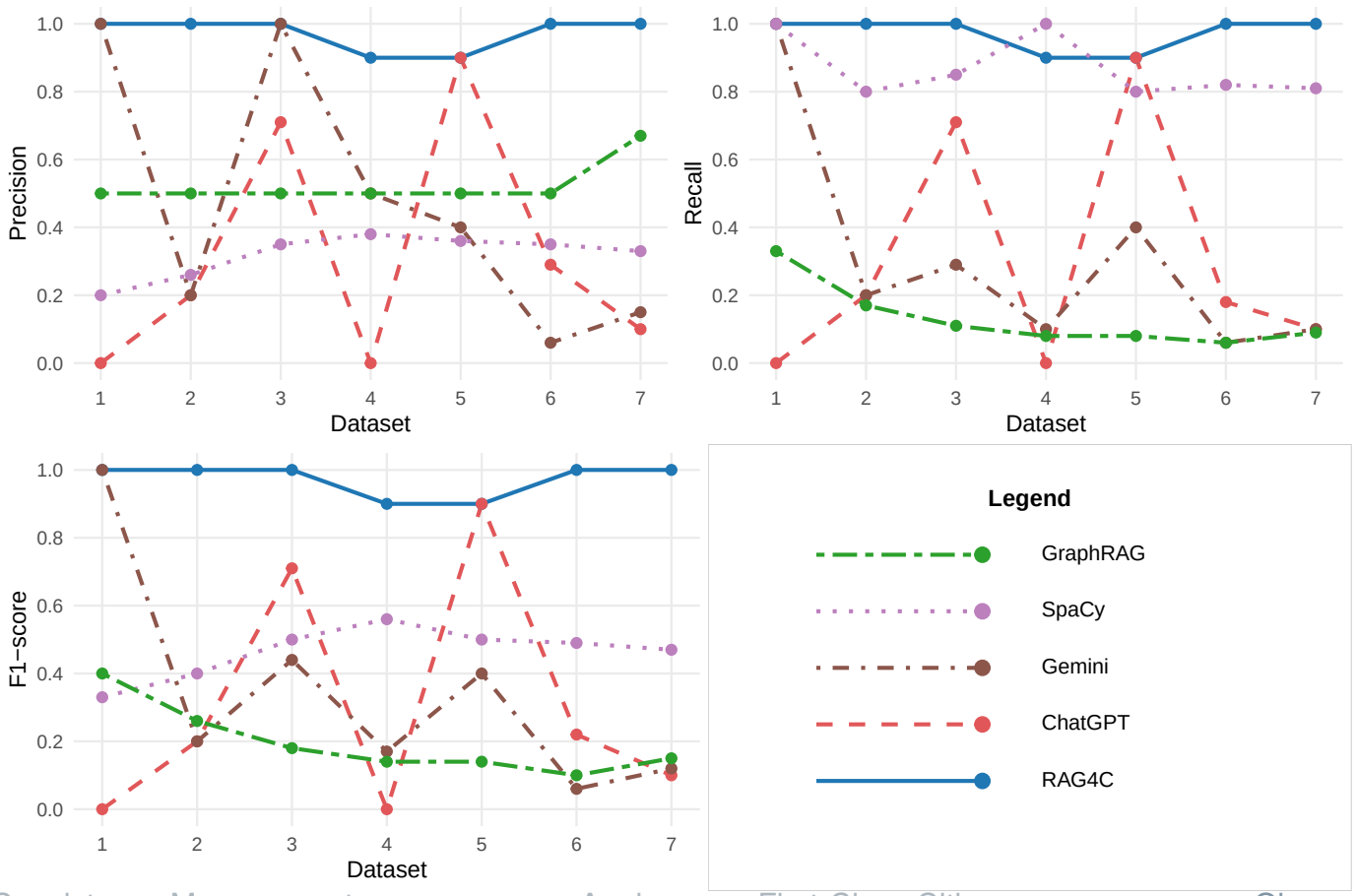
- Transform the extracted graph into a structural model view
- Map `System` entities to classes
- Map observables to attributes
- Map relations to associations

It is not the final design, but helps engineers inspect the extracted systems, observables, and dependencies.

Evaluation setting

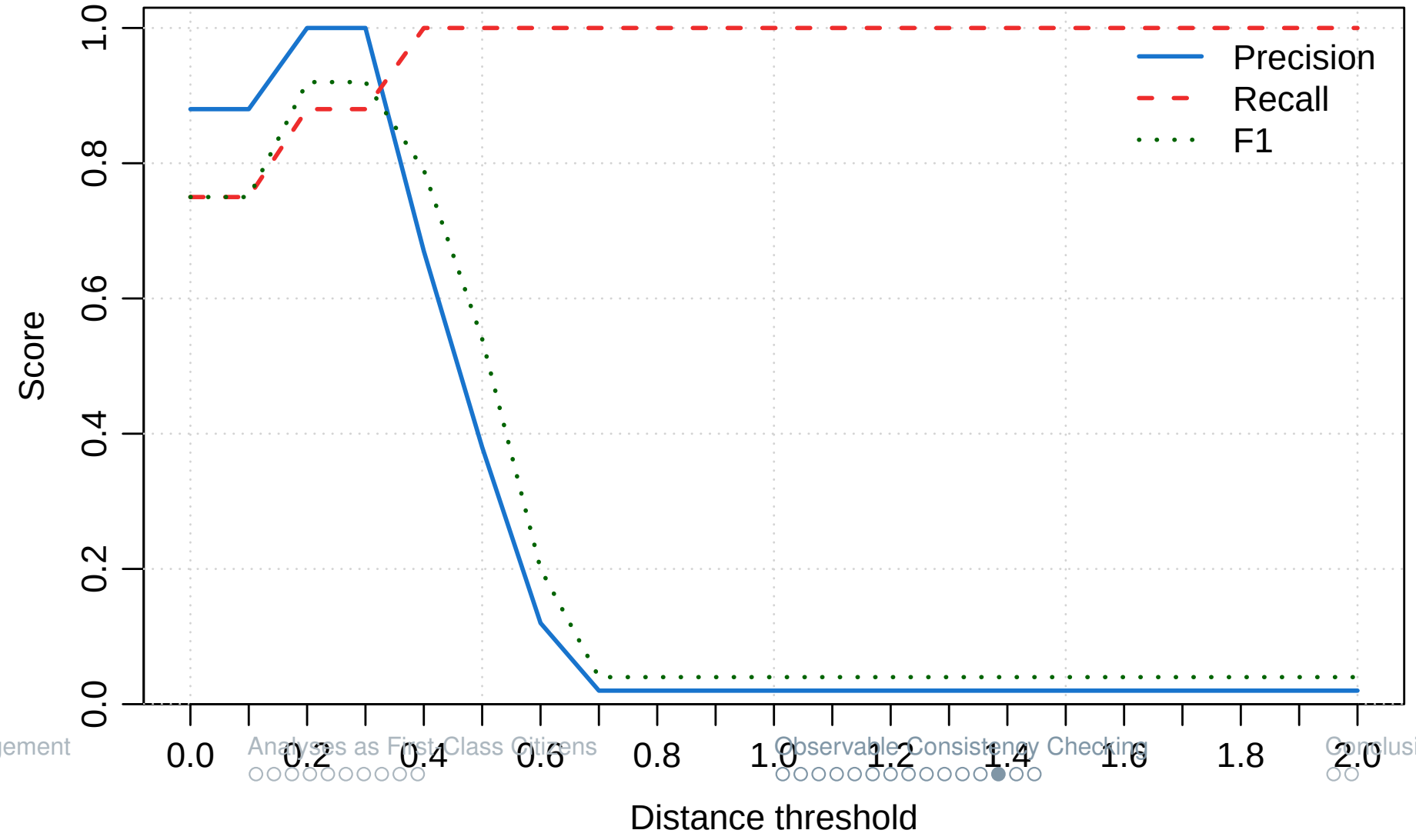
- Synthetic automotive datasets: 7 small human-validated sets + 3 larger scalability sets (A, B, C).
- Industrial requirements: EU project AI$_4$CSM deliverables.
- Baselines: ChatGPT, Gemini, spaCy, and GraphRAG.

Evaluation: observable extraction and synonym groupin



- Observable extraction: macro/average F1 0.99
- Synonym grouping: macro/average F1 0.97

Evaluation: trend of clustering distance threshold



Evaluation: scalability

Data	#Req	Time	Result
A	30	25.3s	UNSAT
B	300	82.7s	UNSAT
C	3000	414.9s	UNSAT
AI_4CSM	151	232.3s	SAT

Most runtime is spent on knowledge-graph construction

$>$ 2 days manual check *V.S.* about 2 minutes automated check + 1 hour manual check.

What have we learned?

Observables
make heterogeneous artefacts comparable.

LLMs
extract candidates, but do not decide on consistency.

Ontologies
make terminology explicit across disciplines.

Solvers
provide evidence: assignments or conflicts.

What are the main contributions?

- Formalisation of analyses and dependencies
- Incremental recertification procedure
- Analysis-aware *V-SUM*
- Ontology-driven extraction of observables and constraints
- Observable consistency across requirements and models

What remains?

- Modular analysis specification language (in progress)
- Unit normalisation and dimensional analysis (in progress)
- Domain-specific trigger function construction (in progress)

Thank you!