Semantics, Validation and Verification of DSLs: An Experience Report

Jozef Hooman, ESI by TNO and RU Nijmegen
Sarmen Keshishzadeh, TU Eindhoven,
Mohammad Mousavi, Halmstad University,
Arjan Mooij, ESI by TNO

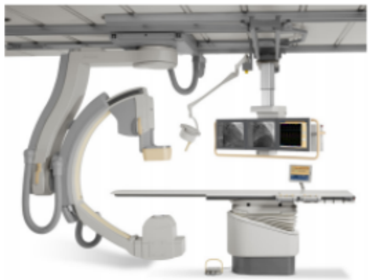
COMMIT/ A public- private research community

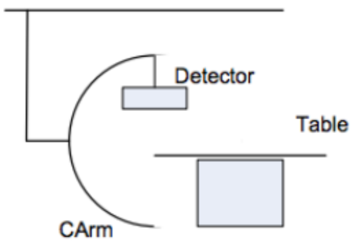
How to pigeonhole me

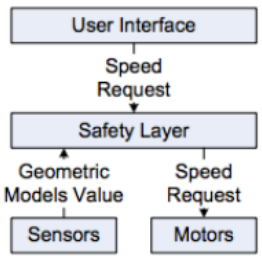
- **2001-2005**: Ph.D., CS, TU Eindhoven with Jan Friso Groote and Gordon Plotkin (Title: Structuring **SOS**)
- **2006-2007:** Postdoc with Luca Aceto (on unaxiomatizability theorems for **process algebras**)
- **2001-2013:** ELE and CS, TU Eindhoven (semantics, process algebra, **model checking**, **testing**)
- **2013-** : CERES, Halmstad University (**model-based testing**)
- **2014-** Secretary of IFIP WG 1.8

Our Domain's Specifics

- Interventional **X-ray scanners**
- Several **moving objects** (tables, arms, detectors) and **restrictions** (e.g., on distances and velocities)
- Layered architecture: **safety layer** to prevent collisions
- Reacting to **users' requests**
- **DSL for early validation, verification and code generation**







Giving it a meaning

- Given a **geometric model value** (from sensors) and an **input speed request** (from user), calculate the set of **active restrictions**, and the **output speed request**
- Determining **active restrictions** depending on geometry and previous state
- Computing output **speed requests** based on **input speed requests** and **most restrictive limits**
- Integrating the semantics in a **symbolic transition system**

Validation and verification

- Model **validation**:
  - against **domain-specific properties**
  - coding the **semantics** and the **properties** in **SMT input** lang.
  - using **Z3** to check for properties

- Implementation **verification**:
  - defining a notion of **conformance**
  - **generating test-cases** from models to check for conformance

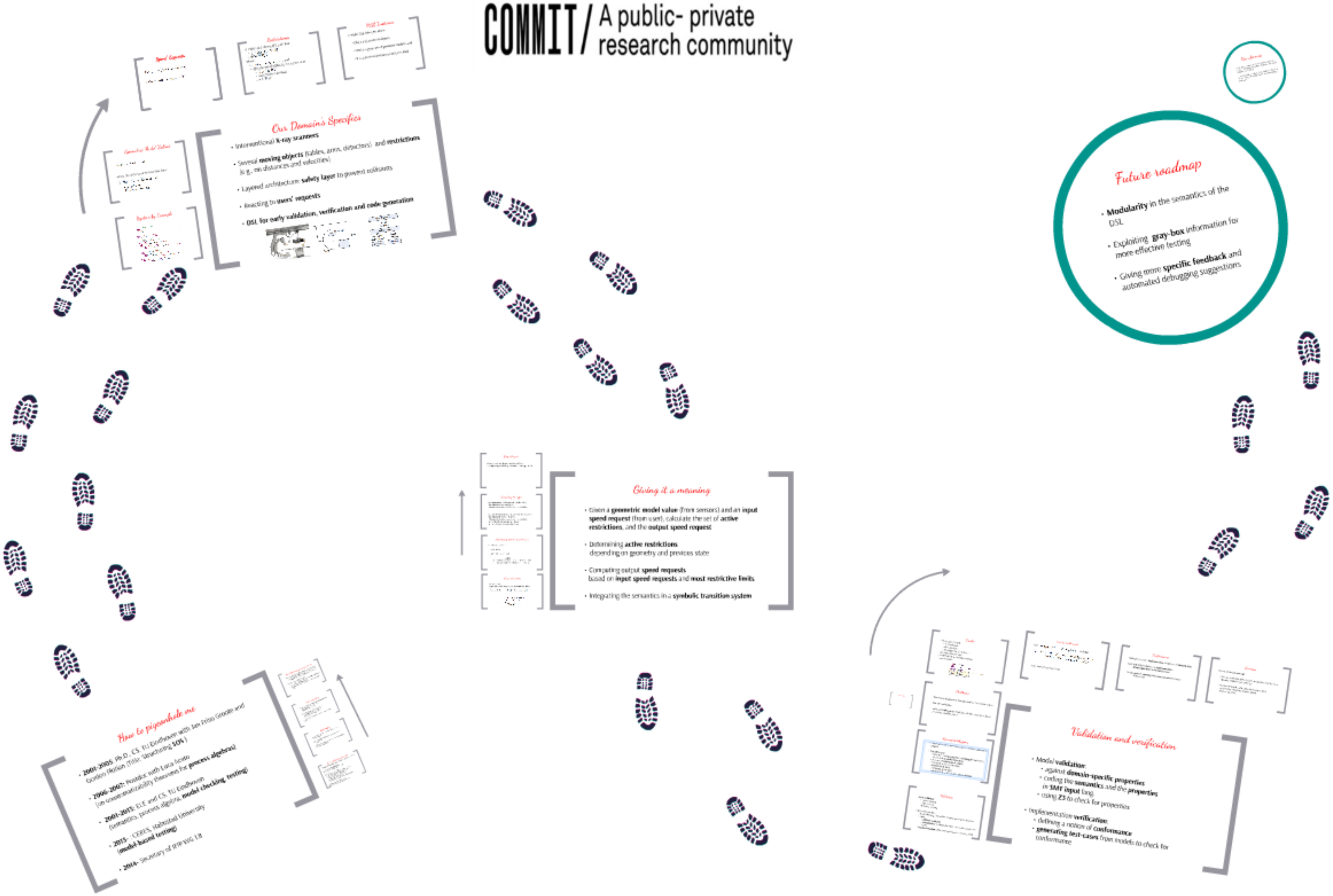
Future roadmap

- **Modularity** in the semantics of the DSL
- Exploiting **gray-box** information for more effective testing
- Giving more **specific feedback** and automated debugging suggestions

Semantics, Validation and Verification of DSLs: An Experience Report

Jozef Hooman, ESI by TNO and RU Nijmegen
Sarmen Keshishzadeh, TU Eindhoven,
Mohammad Mousavi, Halmstad University,
Arjan Mooij, ESI by TNO

COMMIT/ A public- private research community



How to pigeonhole me

- **2001-2005**: Ph.D., CS, TU Eindhoven with Jan Friso Groote and Gordon Plotkin (Title: Structuring **SOS**)
- **2006-2007:** Postdoc with Luca Aceto (on unaxiomatizability theorems for **process algebras**)
- **2001-2013:** ELE and CS, TU Eindhoven (semantics, process algebra, **model checking**, **testing**)
- **2013-** : CERES, Halmstad University (**model-based testing**)
- **2014-** Secretary of IFIP WG 1.8

Structural Operational Semantics

- **Semantical meta-theorems** through **syntactic conditions** on SOS rules

- A yardstick for **language designers**
  (used for substantial languages such as **CIF**)

- Developed theory and tools for:
  - Compositionality (**congruence**) (overview: [TCS'07])
  - **Algebraic properties** (overview: [BEATCS'09])
  - Generating prototype implementations
    (**animators**, **model-checkers**) [SOS'05]
  - Axiomatizations and **equivalence checkers** [SOS'13]

- **Modular semantics** [CONCUR'13] and
  bisimulation for **open systems** [EXPRESS'10]

Model Checking

- Verifying **epistmic** properties of **behavioral** specifications [LPAR'07,TbiLLC'13]
- Exploiting **symmetrical structures** for verifying **actor**-based systems [Acta'10]
- Verifying **sensor network** protocols by identifying **anti-patterns** and avoiding them [iFM'12]

Model-Based Testing

- Applying **conformance testing** to **industrial systems** from:
  - **healthcare** [MoTIP'12,SEFM'13] and
  - **financial** [FSEN'07] domains.
- Extending conformance theories to cater for:
  - **Asynchrony** [SEFM'11, SoSym'14], and
  - **Decompositional testing** [MBT'13]
- Analyzing and reducing complexity of conformance checking [FACS'13]

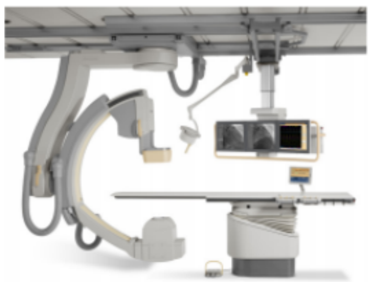
Model-Based Testing (ongoing research)

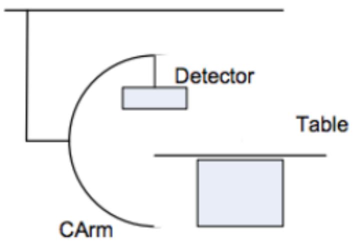
- Applying **conformance testing** to software product lines (SPLs):
  - extending **testing models** and theories to **SPLs** [MBT'14] and
  - **factoring out test-cases for common features** [SAC-SVT'14]

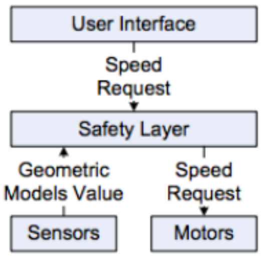
- Model-based **consequence analysis**:
  - Analyzing the possible effects of **non-conforming components** in the overall **system behavior**, and
  - **Diagnosing** the system-level **failures** in terms of **component faults**

Our Domain's Specifics

- Interventional **X-ray scanners**
- Several **moving objects** (tables, arms, detectors) and **restrictions** (e.g., on distances and velocities)
- Layered architecture: **safety layer** to prevent collisions
- Reacting to **users' requests**
- **DSL for early validation, verification and code generation**







Syntax by Example

```
// --- Context Declarations -------
object Table
object CArm
object Detector


model Actuals         predefined
model LookAhead       userdependent

// --- Restrictions -------
restriction ApproachingTableAndCArm
  activation
    Distance[Actuals](Table, CArm) < 35 mm + 15 cm
  effect
    absolute limit CArm[Rotation]
      at ((Distance[Actuals](Table, CArm) - 35 mm) / 15 cm) * 10 dgps


restriction ApproachingTableAndDetector
  activation
    Distance[LookAhead](Table, Detector) < 35 mm + 15 cm
 && Distance[LookAhead](Table, Detector) <
                               Distance[Actuals](Table, Detector)
  effect
    relative limit Detector[Translation]
      at ((Distance[LookAhead](Table, Detector) - 35 mm) / 15 cm)
```

Geometric Model Values

$$GeoVal ::= Mod \rightarrow Dist$$

where *Dist* is the set of distance functions:

$d : Obj \times Obj \rightarrow \mathbb{R}_0^+$ such that

- $d(o_1, o_1) = 0$;
- $d(o_1, o_2) = d(o_2, o_1)$.

Speed Requests

$$MovType == \{Rotation, Translation\}$$

$$SpReq == Obj \times MovType \rightarrow \mathbb{R}^3$$

Restrictions

R: Triples *(act, deact, eff) such that*

- $act, deact \in Cond$
- $eff \in \mathcal{P}(Eff)$

where

- $Cond == GeoVal \rightarrow Bool$ *, and*
- *Eff is the set of triples (lt, om, e) such that*
  - $lt \in \{Abs, Rel\}$
  - $om \in \mathcal{P}(Obj \times MovType)$
  - $e \in Expr$

DSL Instance

A triple (Obj, Mod, R), where

- Obj is a given set of objects,
- Mod is a given set of geometric models, and
- R is a set of restrictions on Obj and Mod

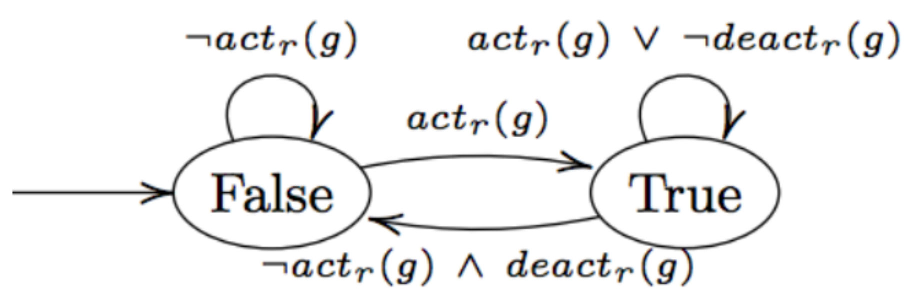
Giving it a meaning

- Given a **geometric model value** (from sensors) and an **input speed request** (from user), calculate the set of **active restrictions**, and the **output speed request**
- Determining **active restrictions** depending on geometry and previous state
- Computing output **speed requests** based on **input speed requests** and **most restrictive limits**
- Integrating the semantics in a **symbolic transition system**

Active restriction

For a restriction r,
a geometric model value g and a current activation state b:

$$CurrAct_r(b, g) = act_r(g) \vee (b \wedge \neg deact_r(g))$$



Determining active restruction set

ASTS(R) = (Q, q, T), where

- Q: R $\rightarrow$ Bool,
- q(r) = false, for each r in R,
- $T \subseteq Q \times Cond \times Q$, where
  - Cond: functions from geo. val. to Boolean expr., and
  - cond(g): conjunction of restrictions active w.r.t. g

Calculating the effect

For a **restriction r**, and a geometric model value g
the effect of r, for each object o:
the **most restrictive** absolute and relative **limits**

For a **set of restictions R** and a geometric model value g
the effect of R, for each object o:
the **most restrictive** absolute and relative **limits**
for each and every active r in R, if any,
or the set of real numbers, otherwise.

Output Request

Given the minimal (absoute and relative) limits,
the output request is the speed vector respecting the limits.

Validation and verification

- Model **validation**:
    - against **domain-specific properties**
    - coding the **semantics** and the **properties** in **SMT input** lang.
    - using **Z3** to check for properties
- Implementation **verification**:
    - defining a notion of **conformance**
    - **generating test-cases** from models to check for conformance

Validation

- **Basic validation**:
  - Syntax checking
  - Type checking
  - Reference chasing

- Advanced validation:
  - Range checking: relative limits in [0,1], non-negative distances
  - Safety
    - **collision avoidance**,
    - **monotonicity**: the closer the objects, the more restrictive the limits
  - **Deadlock freedom**: some user input takes us out of standstill

Automated debugging

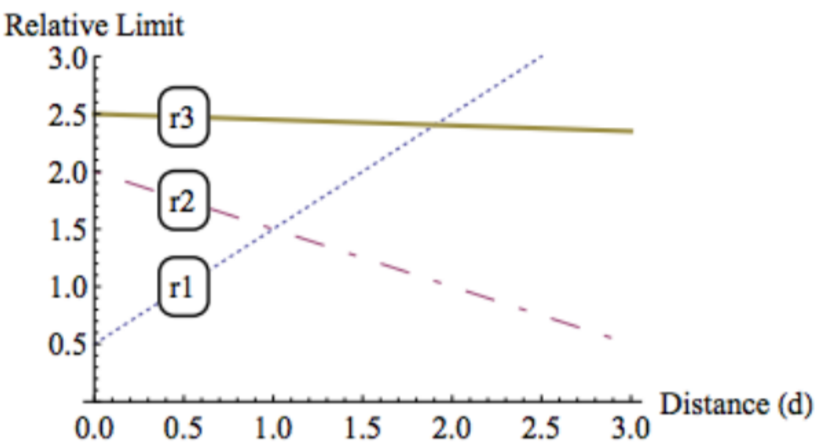
- Pivotal restriction(s): the minimal set of restrictions violating the property

- Delta-debugging:
  - Take two sets of **passing (R+)** and **failing (R-)** restrictions, such that **R- is smaller than of R+**,
  - Pick some **R in between R- and R+**,
  - Validate R (using an SMT solver)
  - If **R fails**, let **R- be R**,
    if **R passes**, let **R+ be R**,
    repeat until R+ and R- differ in **one restriction**

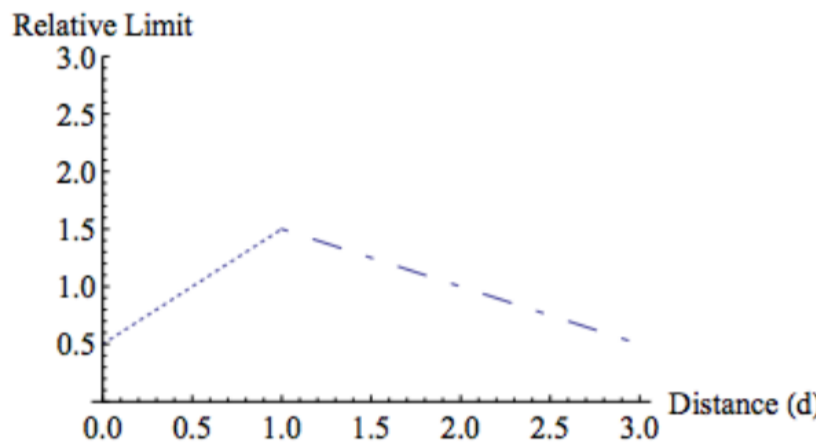
Challenges

- Growing set of geometric functions used by the domain experts
- Masked restrictions
- Technical challenges in translation and SMT solving (non-linear constraints, quantification)

Masked restrictions



(a) Effects of restrictions



(b) Unmasked parts of restrictions

Results

- On an actual model:
  - 16 parameters,
  - 81 restrictions,
  - 264 properties (5 patterns)
- 226 violations detected,
- traced back to a handful restrictions
- Took 2 minutes

```
restriction ApproachingTableAndCArm
  activation
    Distance[Actuals](Table, CArm) < 35 mm + 15 cm
  effect
    absolute limit CArm[Rotation]
      at ((Distance[Actuals](Table, CArm) - 35 mm) / 15 cm) * 10 dgps
```

Multiple markers at this line
- Potential (Rotation) deadlock for CArm
- Absolute limit for CArm rotation should be non-negative

Traces and runs

Run: $q_0(g_1, ur_1, or_1)\ldots(g_n, ur_n, or_n)q_n$ such that

$$(q_0 = q) \wedge \forall i \in \{1, \ldots, n\} \; . \; (or_i = [\![output_R]\!]_{L(q_i)}(g_i, ur_i)) \wedge \\ (\exists cond \in Cond.(q_{i-1}, cond, q_i) \in T \wedge cond(g_i))$$

Trace: any finite prefix of a run

Conformance

Testing assumption: **implementation** behaves as an **unknown ASTS**

Goal: to establish whether the **implementation** is
(trace) equivalent to the **DSL instance**

Means: generate **covering test-cases** (concrete executions)
from the DSL

Coverage

The two notions of coverage:

- For each reachable symbolic state, an execution hits the state (generic, similar to node coverage)

- For each reachable state, and each reachable limit, an execution checks for its enforcement (domain-specific)

Future roadmap

- **Modularity** in the semantics of the DSL
- Exploiting **gray-box** information for more effective testing
- Giving more **specific feedback** and automated debugging suggestions

Key references

- A.J. Mooij, J. Hooman, R. Albers: Early Fault Detection Using Design Models for Collision Prevention in Medical Equipment. FHIES 2013.

- S. Keshishzadeh, A.J. Mooij, MRM: Early Fault Detection in DSLs Using SMT Solving and Automated Debugging. SEFM 2013.