Integrated Specification, Validation and Verification with HybridUML and OCL

applied to the BART Case Study

Stefan Bisanz, Paul Ziemann, and Arne Lindow

Universität Bremen

Germany

{bisanz,ziemann,lindow}@informatik.uni-bremen.de

Outline

1. Ingredients: HybridUML and OCL
2. Specification by Example – the BART Case Study
3. Simulation Semantics (a sketch)
4. Integrated Validation/Verification
5. Conclusion

Ingredient 1: HybridUML – Specification Formalism for Hybrid Systems

- Formal Semantics

- UML

Ingredient 1: HybridUML – Specification Formalism for Hybrid Systems

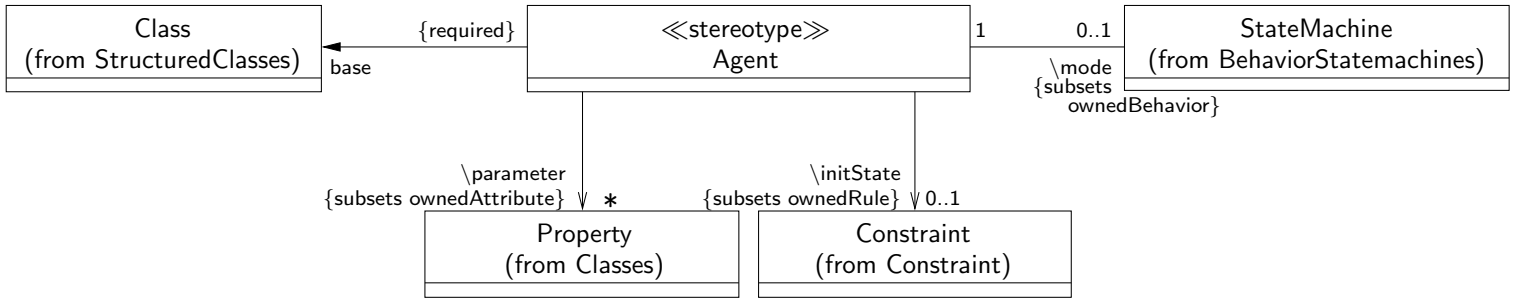
- Formal Semantics
  - \+ Hybrid automata (Henzinger)
  - \+ Hierarchic Modelling
  - \+ Semantics by Transformation $\Rightarrow HL^3$
  - \+ Semantically well-defined
  - \+ Executable in Hard Real-Time
- UML

Ingredient 1: HybridUML – Specification Formalism for Hybrid Systems

- Formal Semantics

- UML

  + Customisation of UML 2.0 – HybridUML Profile:



  Additional constraints, i.e. `self.mode->forAll(oclIsTypeOf(Mode))`

  + Enhanced by (time-continuous) Flow Constraints and Invariants

Ingredient 2: OCL – The USE approach for validation/verification

OCL

- UML's standard expression language
- for specifying Constraints; particularly: Constraints on System States

USE

- USE – Tool for checking Constraints wrt. System State
- provides Class Diagrams, Object Diagrams, OCL Constraints
- Separate Scripting Language providing "snap shots" of System States (i.e. a Set of Objects) corresponding to Class Diagrams
- are calculated by Pascal-like Procedures based on OCL

Our Approach: Ingredient 1 + 2

Benefits:

- Accurate Modeling by use of Continuous Time
- Behavioral Specification *by means of* UML Model
- Behavioral Specification *is Part of* UML Model – one single (UML) Model
- Integration of Validation/Verification features into Simulation

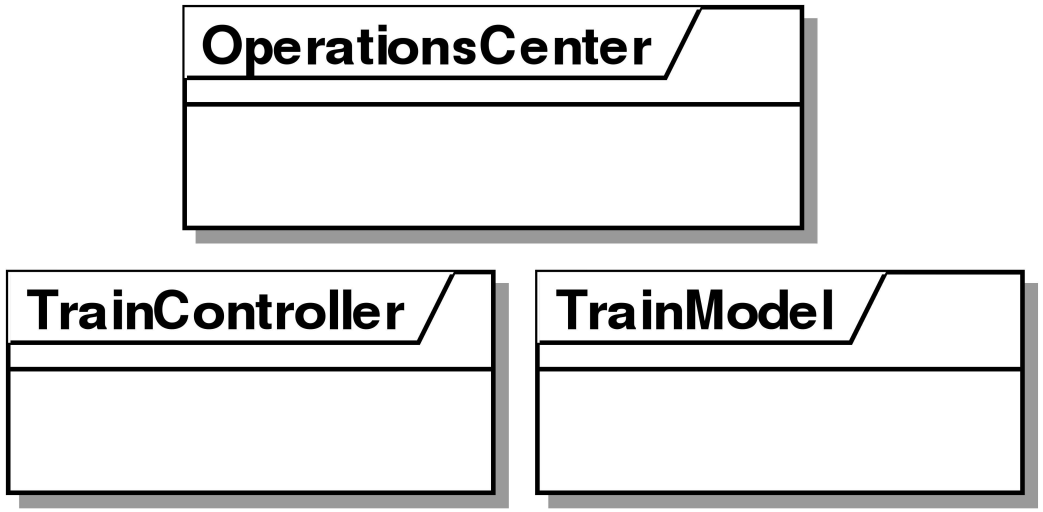
Specification by Example – the BART Case Study

- Advanced Automated Train Control (AATC)
- part of Bay Area Rapid Transit system (BART)
- Main Objective: Controlling Velocity and Acceleration of Trains
- of course – Hazards should be avoided

TrainModel Physical Environment.

OperationsCenter Central Part of Computer System, computes Velocities and Accelerations for all Trains.

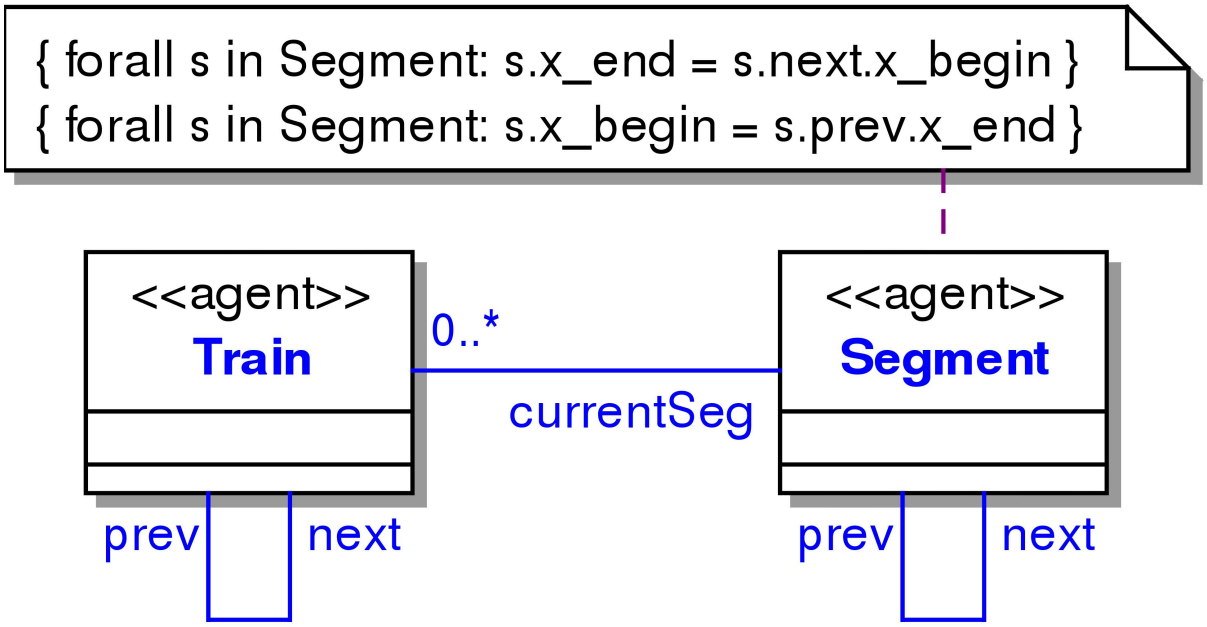
TrainController Local Controller per Train.



Packages of the BART model.

Simplified Environment:

- Trains travel on a single circular Track.
- Track is divided into Segments.
- Segments may be (temporarily) closed at the end.
- Segments have a Civil Speed.



Physical Train Model.

OperationsCenter Composite Agent consisting of RemoteTrainControllers.

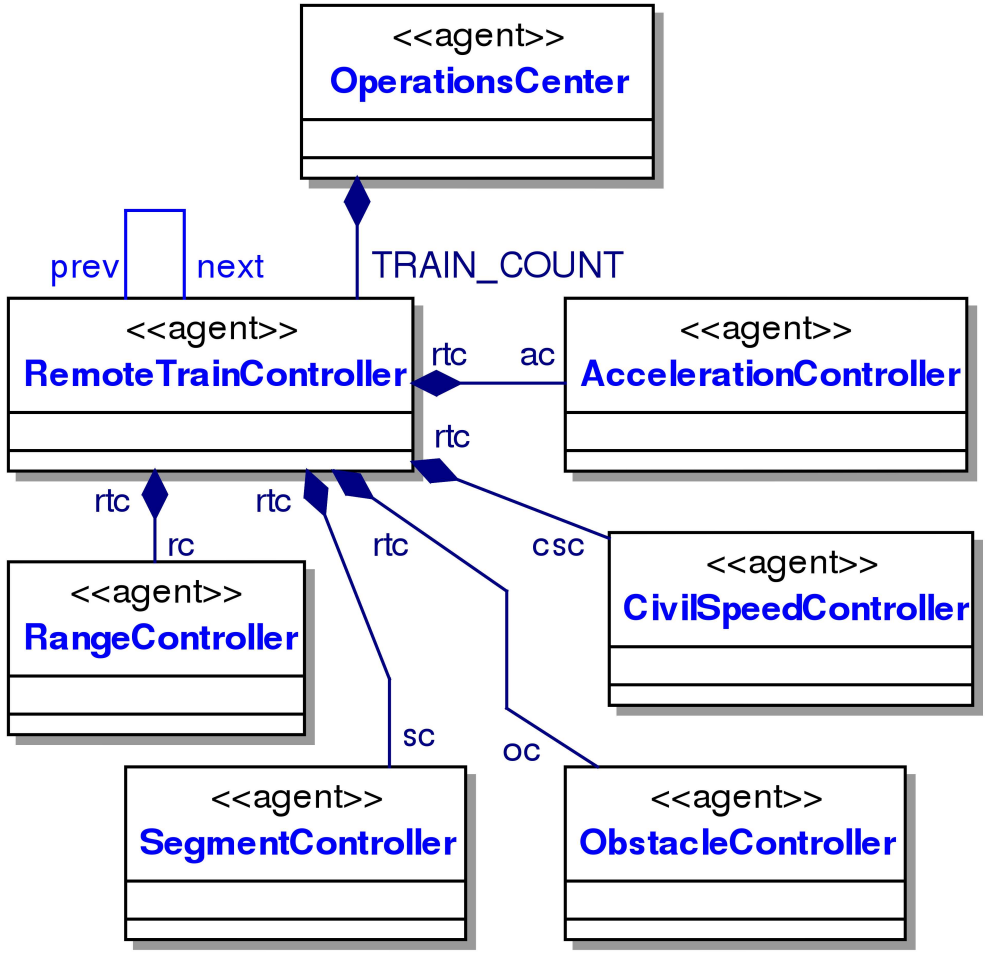
RemoteTrainController Composite Agent consisting of Basic Agents.

RangeController Basic Agent continuously calculating a train's horizon.

SegmentController Calculates Segments in Range, along with their Civil Speeds.

CivilSpeedController Provides the Acceleration needed to reach the lowest Civil Speed in Range.

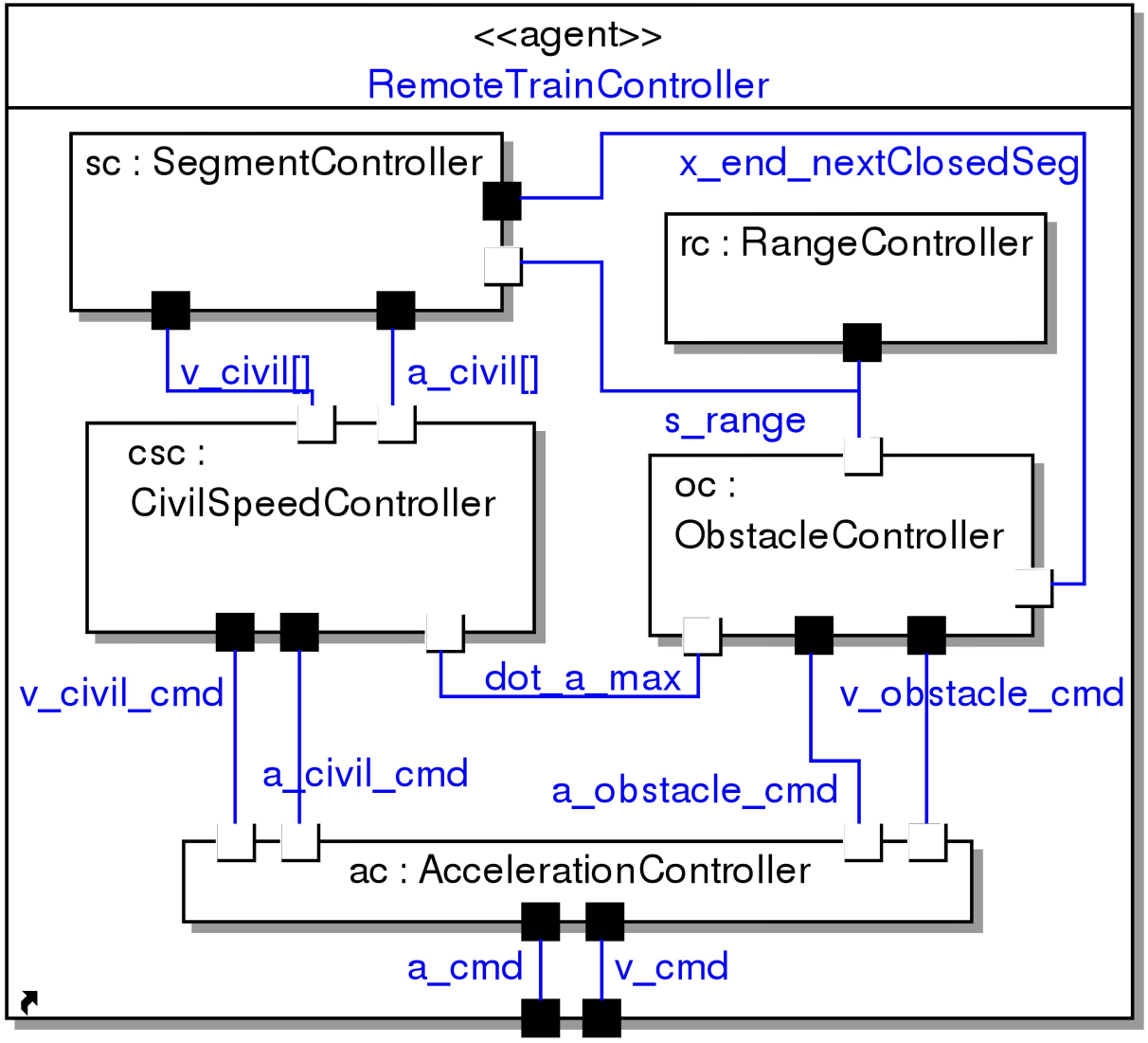
...



Operations Center.

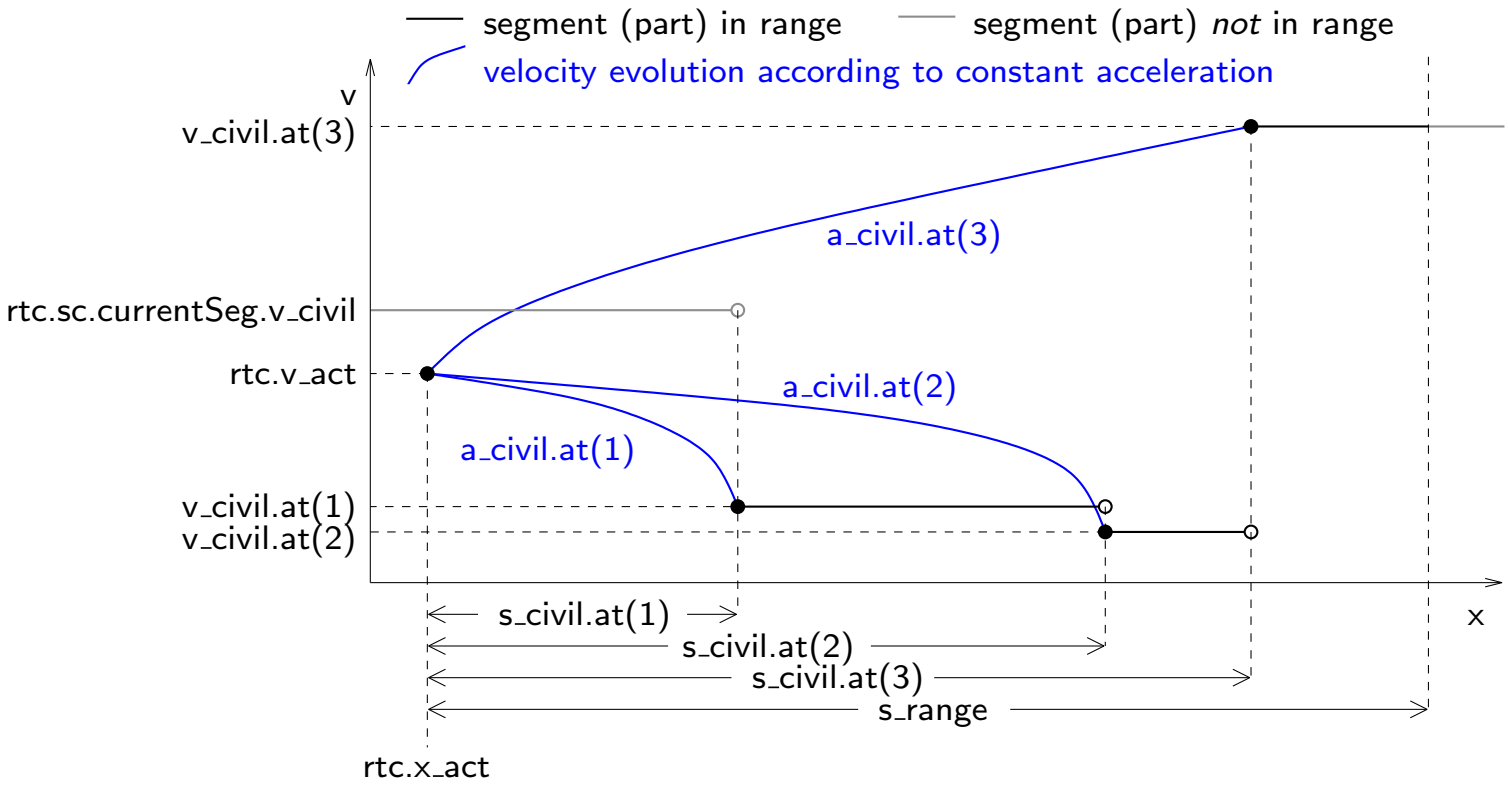
Composite Structure view:

- Detailed View on an Agent's Structure.
- Ports and Connectors define how Variables (and Signals) are shared.
- Read/Write (black) and Read-Only Access (white).



Structure of RemoteTrainController.

Civil Speed Controller's intended Behavior:



- Chooses the Lowest Acceleration for Segments in Range.

- Additionally limits Acceleration wrt. Current Civil Speed.

Behavior Specification:

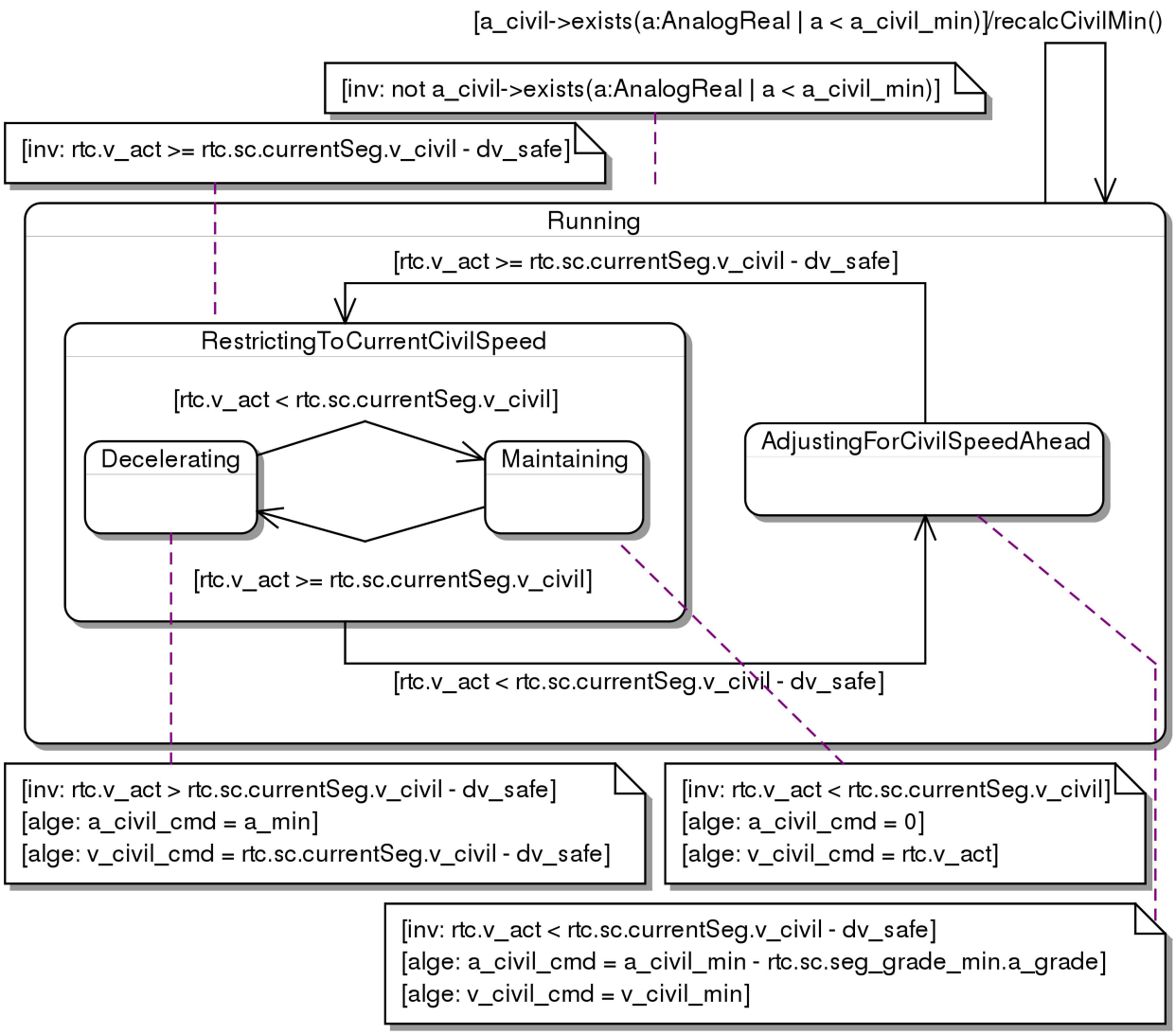
Hierachical State Machines

alge/flow Defines a Time-Continuous Calculation activated by its *Mode*.

inv Defines an Invariant Condition that must hold for its *Mode* to be active.

Non-urgent Transitions

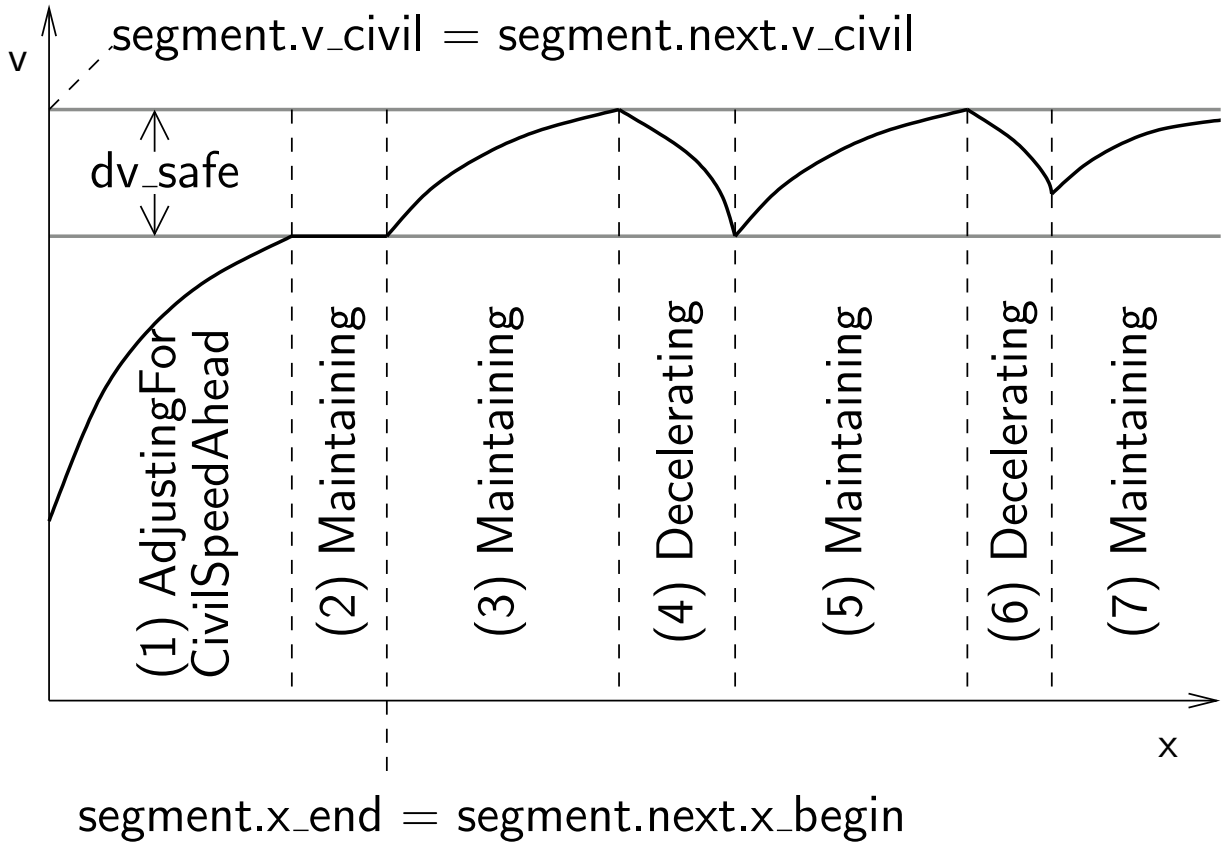
⇒ Invariant Conditions and Transition Guards model "reaction ranges"



Behavior of CivilSpeedController.

Scenario for
RestrictingToCurrentCivilSpeed:

- 2 Segments
- Identical Civil Speed
- Different Grades:
  segment.a_grade = 0 and
  segment.next.a_grade > 0



Illustrated functionality of RestrictingToCurrentCivilSpeed.

Simulation Semantics

HybridUML: interleaving of discrete and continuous steps

Discrete step
- A single basic agent fires one transition.
  - No time passes.

Continuous step
- All agents synchronously let time pass.
  - Active algebraic and flow conditions are applied.
  - All invariants of current mode configurations must be satisfied.

1. Active invariants are checked. $\Rightarrow e(c)$: Continuous step c admissible?

2. Set $T_{enabled}$ of enabled transitions is calculated.

3. $|T_{enabled}| > 0 \Rightarrow$ One enabled transition $t \in T_{enabled}$ is chosen non-deterministically.

4. $e(c) \wedge \neg(|T_{enabled}| > 0)$ — c is chosen.

   $\neg e(c) \wedge |T_{enabled}| > 0$ — t is chosen.

   $e(c) \wedge (|T_{enabled}| > 0)$ — Non-deterministic choice of t or c.

   $\neg e(c) \wedge \neg(|T_{enabled}| > 0)$ — Deadlock.

5. The chosen step is taken, then proceeded with 1.

Integrated Validation/Verification

Verification is the process of determining whether the output of a lifecycle phase fulfils the requirements specified by the previous phase.

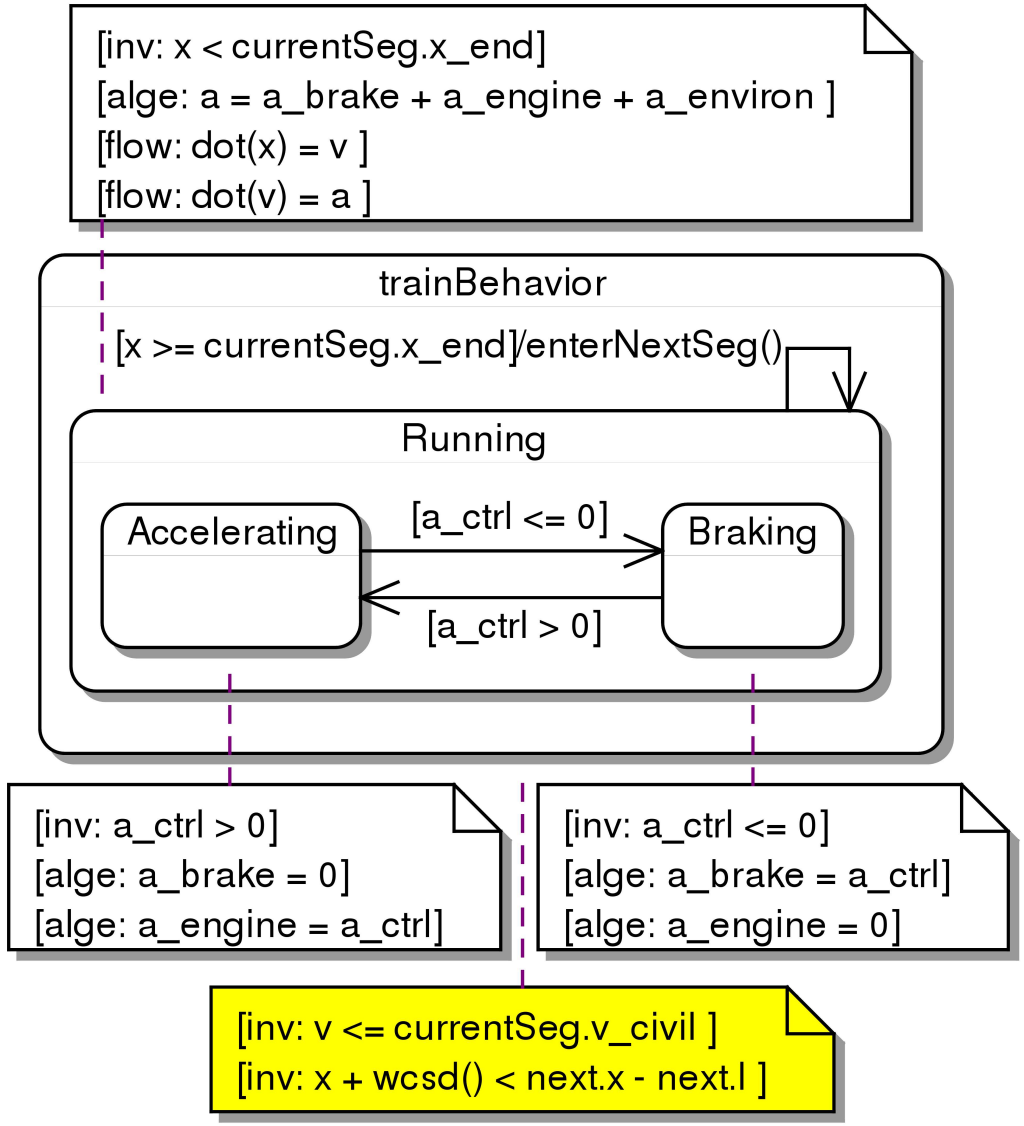
Validation is the process of confirming that the specification of a phase, or of the complete system, is appropriate [...]

Top-Level Invariant Constraints – (can) represent System Requirements (here: Safety Requirements)

1. A Train's velocity never exceeds the current Civil Speed.

2. A Train never touches the Train in front.

Safety Requirements specified as Top-Level Invariant Constraints:



Expected Behavior of the Physical Train.

When Deadlock in the Simulation Loop is observed:

- A violated Invariant exists.
- No enabled Transitions exist.

USE Tool then assists:

- Evaluates active Invariants and Transition Guards.
- Shows the responsible Part(s) of violated Constraint(s).

... Faulty Design $\Rightarrow$ *Verification* fails, but ...

Special Case: A Top-Level Invariant (representing a System Requirement) is violated.

- System State does not satisfy System Requirement.
- System Requirements shall never be violated, independent of Transition Guards.
- Verification vs. Validation – a matter of interpretation:
  - Assumption: The Invariant (Requirement) is correct. $\Rightarrow$ *Verification* fails.
  - Assumption: The System State is correct. $\Rightarrow$ *Validation* fails.

Conclusion

so far: Combination of HybridUML and OCL features

- Semantically well-founded Hybrid Simulation
- Embedded Verification/Validation
- $\Rightarrow$ Testing with Random Test Data Generation

future work (among others):

- Investigation of more elaborate Test Data Generation Algorithms
- Extension of Verification/Validation Concept to e.g. Livelocks

The end.