Extendable Toolchain for Automatic Compatibility Checks

OCL @ MODELS 2016

Vincent Bertram, Alexander Roth, Bernhard Rumpe, Michael von Wenckstern

Software Engineering
RWTH Aachen
http://www.se-rwth.de/

Outline

1. Motivation and Example
2. Compatibility and Industrial Requirements
3. Introduction Extendable Toolchain
4. Conclusion

Motivation - General

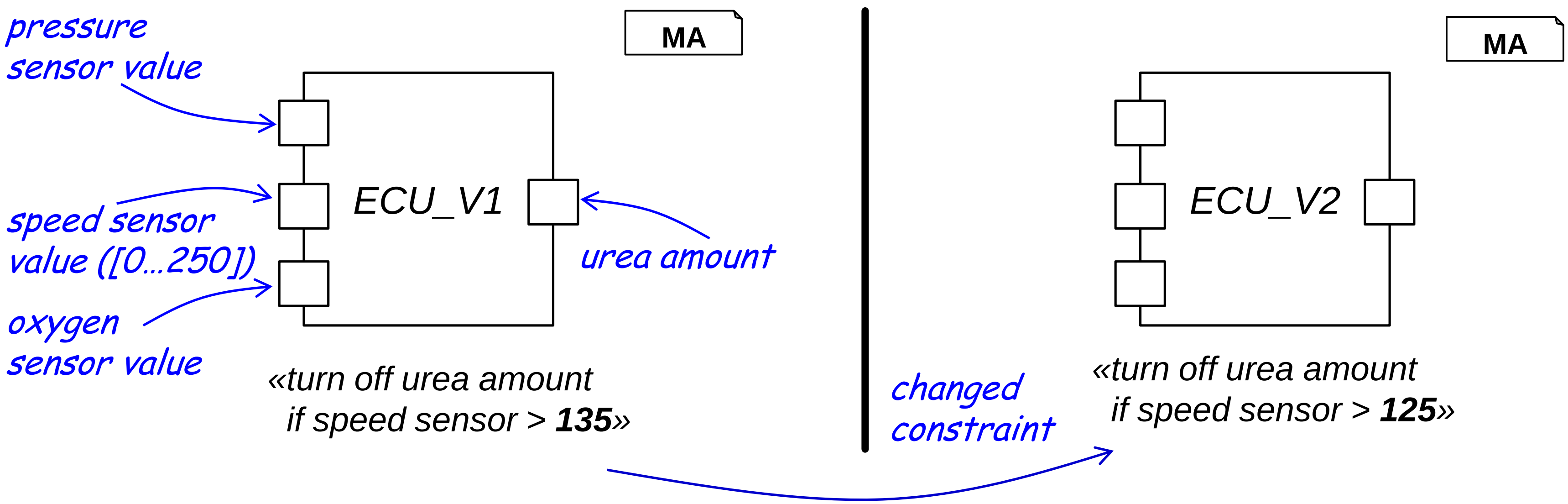
- Automotive software has a dramatically increased number of software components
  - Example: Emissions control systems
  - They have complex systems with different HW / SW components
  - Various tools are used inside a development toolchain

- Vehicles will be continually improved
  - Existence of evolution and variants of function components
  - As well as large and complex product line families

- Safety of software components is very important in many areas (esp. <u>automotive</u> / aerospace / railway industry)
  - Safety-relevant software in the sense of ISO 26262
  - Automotive Safety Integrity Level (ASIL) classification

Motivation - Example

Automotive Emission Control System (simplified)



A developer team member is unsure if the new version can be used in the US and in Germany.

<u>Varying regulations in different countries:</u>
In Germany the emission control can be turned off if the speed is greater 120 km/h, whereas in the US it can be turned off if speed is greater 128 km/h.

Outline

Structural Compatibility

- Component compatibility: Structural compatibility serves as a first indicator as it is an important prerequisite for full compatibility, which would also enclose behavioral compatibility.

- Compatibility of different versions and variants for function components

  - **V2 + V1:** V2 is backward compatible, V2 can replace V1.

  - **V2 – V1:** V2 is forward compatible, V2 can be replaced by V1.

  - **V2 0 V1:** V2 is full compatible to V1, both components can replace each other (have exactly the same behavior)

Requirements from Industry[1]

(1) Compatibility constraints should be defined in comprehensive and concise notation

(2) Method should support heterogeneous C&C architecture models

(3) Developers should be able to modify structural compatibility constraints at runtime

(4) Meaningful and model related error messages for engineers

(5) Genuine C&C model files should not be modified

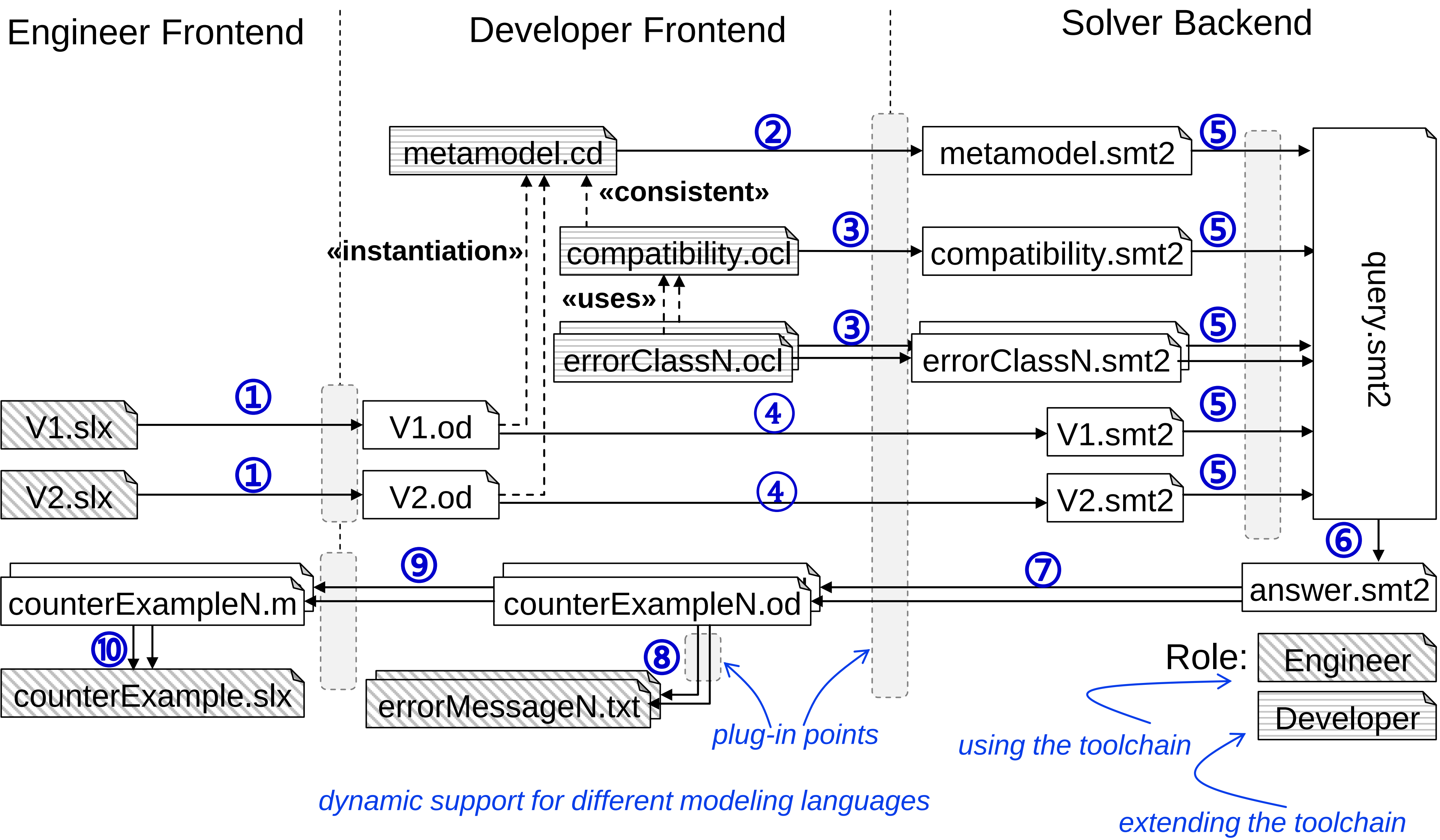
(6) Compatibility checking should be easy for engineers

[1]http://spes2020.informatik.tu-muenchen.de/spes_xt-home.html

Outline

1. Motivation and Example
2. Compatibility and Industrial Requirements
3. Introduction Extendable Toolchain
4. Conclusion

Extendable Tool Chain - Overview



Vincent Bertram

Software Engineering
Chair (Bernhard Rumpe)

RWTH Aachen University

Syntax example of Z3 and OCL/P

Z3

```
(define-fun IsIn_Number_Range((v Number) (r Range)) Bool
   (and (GreaterThen_Number_Number v (minimum r))
        (LessThen_Number_Number v (maximum r))
        (or (not (resDefined r))
            (Equals_Number_Number
                (Mod_Number_Number (Minus_Number_Number v (minimum r))
                                   (resolution r))
                (mk-number 0) ))))
```

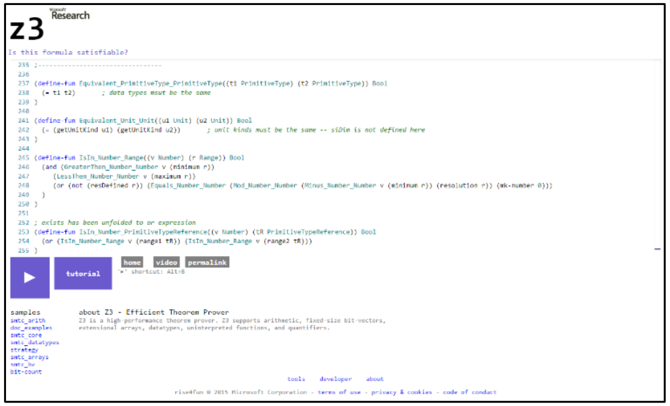
OCL/P

```
def boolean infix (Number v) in (Range r) is:
   result = v >= r.min && v <= r.max &&
      (~r.res || (v - range.min) % range.res == 0)
```

OCL/P has a better understandable mathematical infix notation, while Z3 uses a parenthesized prefix notation which is not easy to read and write.

A more complex example, comparing two ADAS, and showing how generated SMT code actually looks like is online available.

http://rise4fun.com/Z3/2AsLg



Vincent Bertram

Software Engineering
Chair (Bernhard Rumpe)

RWTH Aachen University

Simulation (Preorder) Algorithm

model	**time [s]**	**time* [s]**	**change in generated Z3 code**
m1	timeout	10.08	-
m2	126.68	10.44	remove custom datatypes
m3	93.55	12.86	change encoding of meta-model
m4	138.38	10.47	use `ite` (if-then-else) instead of `implies` after quantifier
m5	70.74	8.34	replace enumeration datatypes by integers
m6	19.05	4.33	replace id hash with an unique id starting at zero
m7	15.17	4.23	remove unnecessary `ite`s when translating OCL to Z3

* added `simplify solve-eqs smt` as solver strategy

Impact of generated SMT code on Z3's execution time (A = 126 / B = 96)

Z3 …

```
; meta-model definition
(declare-datatypes () ((Connector (mk-connector (source (List Name))
   (target (List Name)) (id ID)))))
; instance creation
(mk-connector (insert n_switch1 (insert n_out1 nil))
  (insert n_mul (insert n_in2 nil)) id_1593458942)
```

Z3 …

```
(define-fun getConnectorSourceFromId ((id Int)) (List Int)
(declare-datatypes () ((Connector (mk-connector (source (List Name))
   (ite (= id 2) (insert 2 (insert 56 nil))
   (ite (= id 14) (insert 0 (insert 56 nil))
```

Z3 code used in first version (top) and last version (bottom)

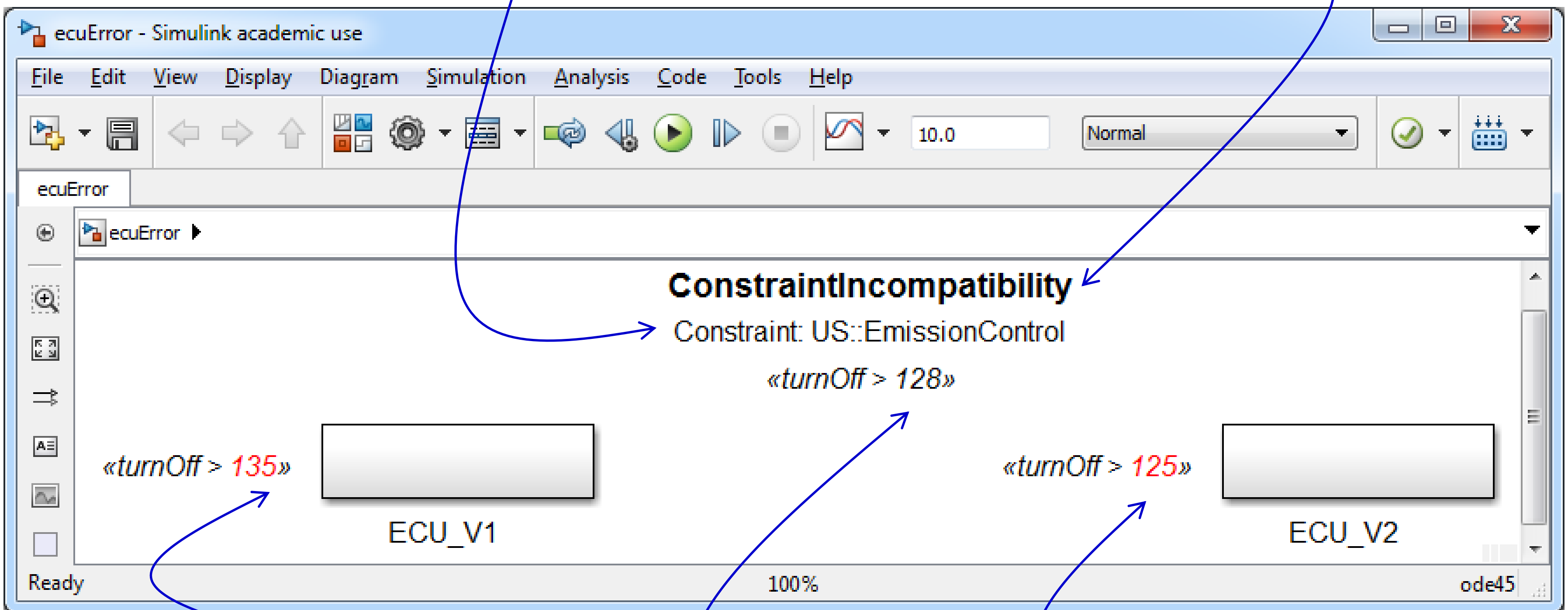
Vincent Bertram
Software Engineering Chair (Bernhard Rumpe)
RWTH Aachen University

Counter Example as Simulink Model

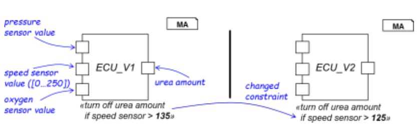
violated constraint

type of incompatibility

Engineer Fronted



backward compatibility
of ECU V2 to ECU V1



provided counterexample

as no counterexample for EU::EmissionCOntrol is provided this constraint is not violated

Vincent Bertram

Software Engineering
Chair (Bernhard Rumpe)

RWTH Aachen University

Outline

Conclusion

- Updates of software components are very unpredictable due to
  - different versions
  - variants
  - and configuration options

- Presentation of a highly adaptable infrastructure to check compatibility constraints
  - based on a generic meta-model and employs OCL at runtime
  - customizability is achieved via plug-in points
  - different views for developer and engineer are given inside the presented toolchain
  - since all transformations are dynamically executed during the checking process, redefinitions and extensions of compatibility definitions and compatibility variations (e.g. for local markets) are supported

Conclusion (Requirements from Industry)

(1) Compatibility constraints should be defined in comprehensive and concise notation

- Usage of OCL/P instead of plain solver code as it is easer to read and understand
- Feasible, not too formal for the developer
- Introduction of two user types (engineer and developer)

(2) Method should support heterogeneous C&C architecture models

- Plug-in structure for use of different modelling languages and solvers
- Trough own meta-model and plugin structure it is usable for further modeling languages as the meta-model is based on an intensive analysis of well established modeling languages.

Conclusion (Requirements from Industry)

(3) Developers should be able to modify structural compatibility constraints at runtime

- OCL constraints can be added dynamically
- 63 constraints have been identified

(4) Meaningful and model related error messages for engineers

- Textual / graphical results instead of sat / unsat
- Constructs counter-example if not similar

(5) Genuine C&C model files should not be modified

- New m-files are generated instead of changing the original ones.
- Textual results presented in individual files

Thank you for your attention.