34. The Meta-CASE-Tool (e)MOFLON

A Meta-CASE tool and a 1-TS-Software Factory

Prof. Dr. Uwe Aßmann

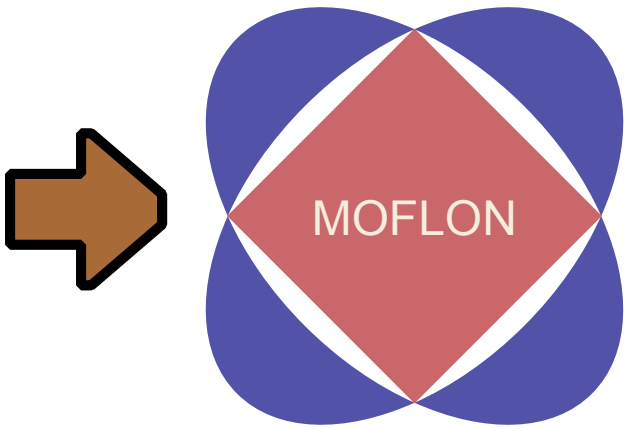
Technische Universität Dresden

Institut für Software- und Multimediatechnik

http://st.inf.tu-dresden.de

Version 19-0.6, 24.01.20

1) MOFLON Meta-CASE-Werkzeug
2) Architecture
3) TGG



DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Reading

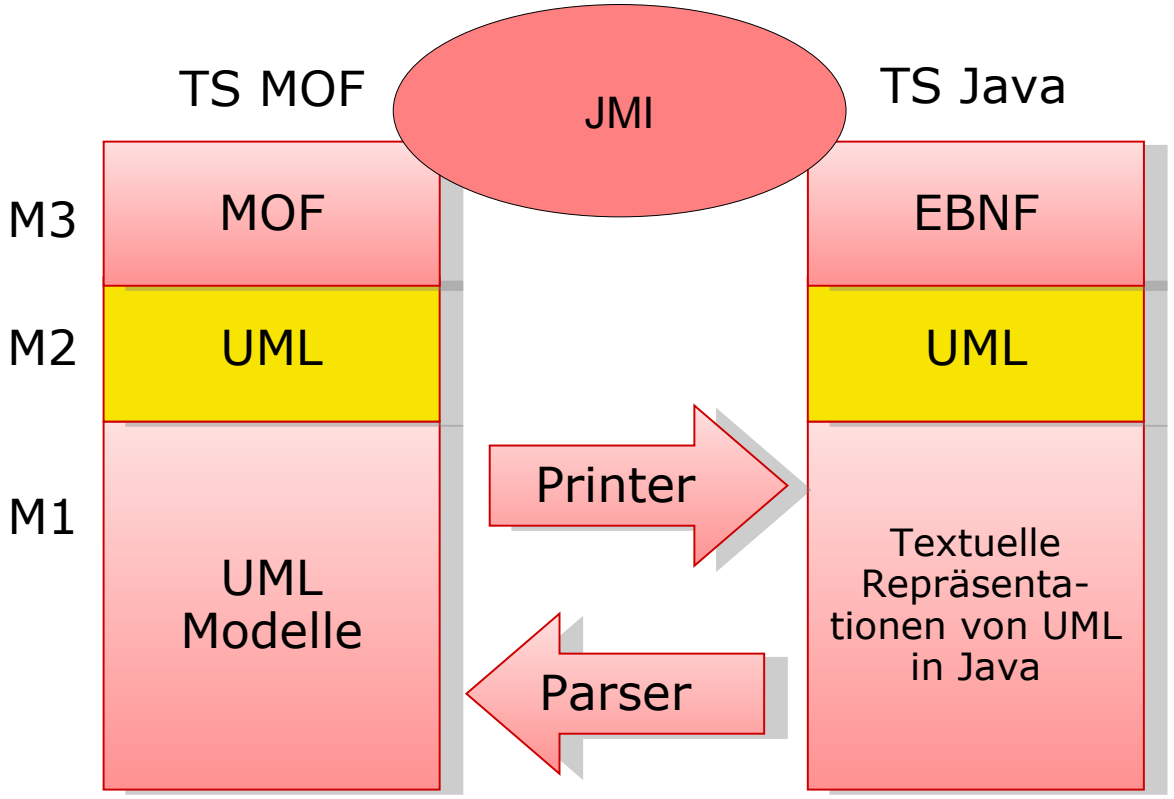
- MOFLON Website http://www.moflon.org
- The Eclipse-Version of the tool is called eMOFLON
  - eMOFLON tutorial
  - http://www.moflon.org/fileadmin/download/moflon-ide/eclipse-plugin/documents/release/eMoflonTutorial.pdf
- A Comparison of ATL and Story-Driven Modeling (Fujaba-style GRS)
  - http://www.es.tu-darmstadt.de/fileadmin/download/publications/spatzina/PP_AGTIVE_2011.pdf
- MOFLON Training
  - http://moflon.org/documentation/links.html
- MOFLON Tutorial
  - http://moflon.org/documentation/tutorial.html

34.1. eMOFLON Introduction

- MOFLON is a Metamodelling Toolset (Meta-CASE tool) of TU Darmstadt, Fachgruppe Real-Time Systems, Prof. Andy Schürr
  - MOFLON uses OCL (logic) for the checking of wellformedness of all models
  - MOFLON is an extension of Fujaba offering graph rewriting www.fujaba.de
  - MOFLON supports Triple Graph Grammars (TGG, see ST-II)
- eMOFLON supports the Technical Space of E(MOF)
  - OCL 2.0
  - JMI 1.4
  - XMI 2.1
- eMOFLON relies on metamodel composition of MOF, OCL
- eMOFLON relies on metamodel mappings between MOF, OCL, XML and Java

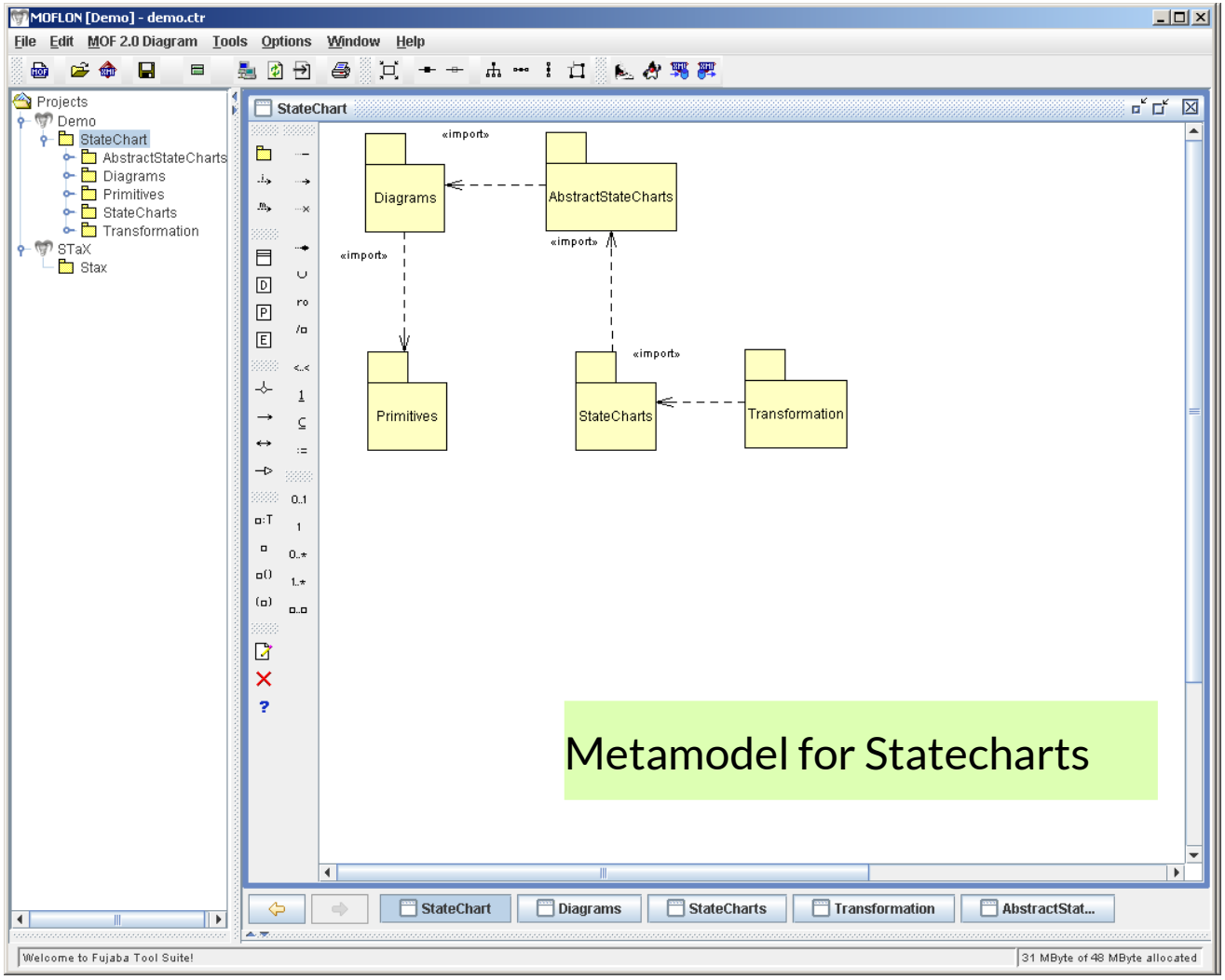
Code Generation with JMI, transformative TS-Bridge for (E)MOF and Java for the Language UML

- Java Metadata Interchange (JMI) is similar to XMI, a TS bridge between (E)MoF and Grammarware
- Only for UML available (language mapping)

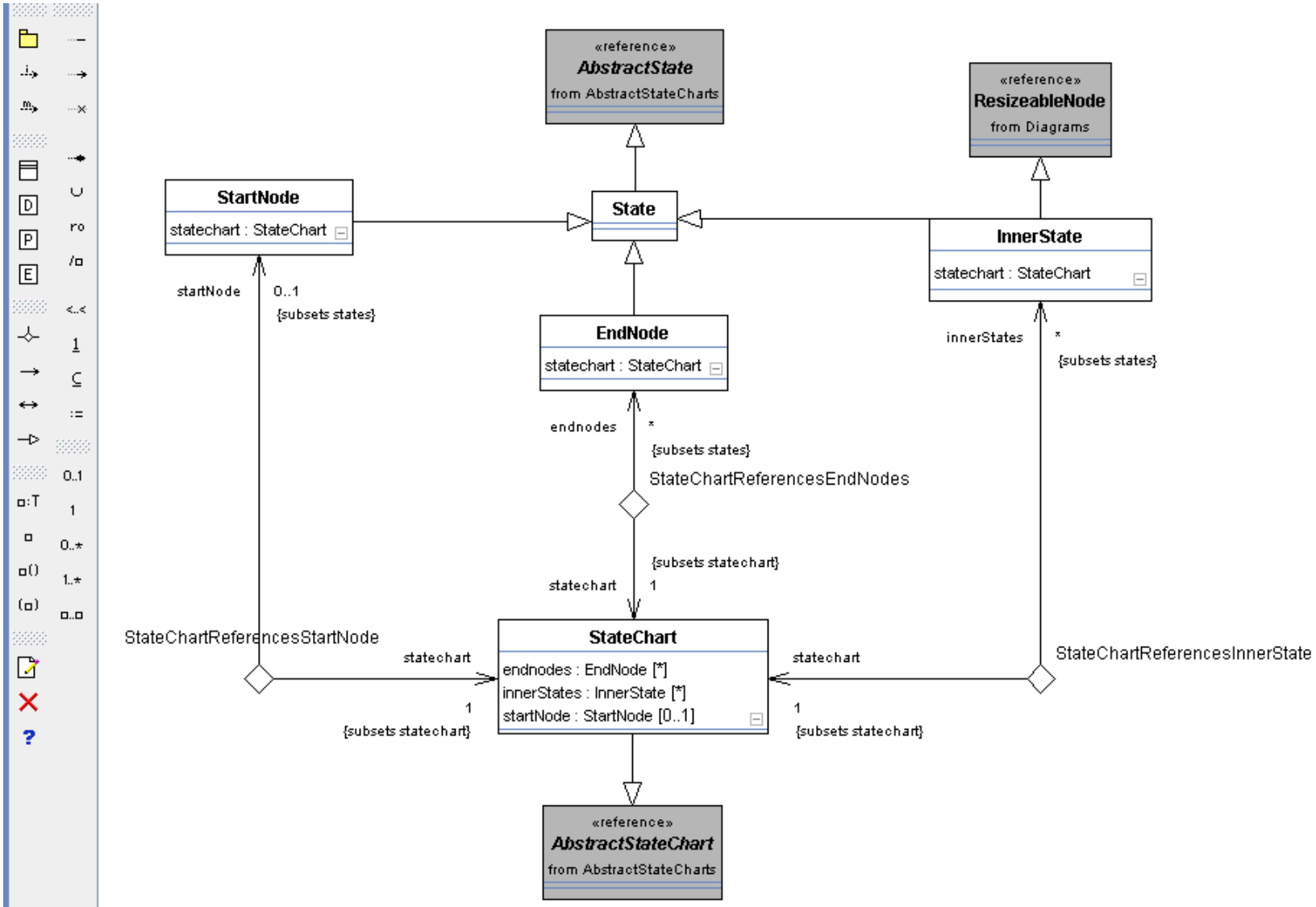


(e)MOFLON Example 1: Metamodel for Statecharts: Development Process

1) Create metamodel
2) Generate Code (Generate repository with constraint-checker) with the JMI interfaces

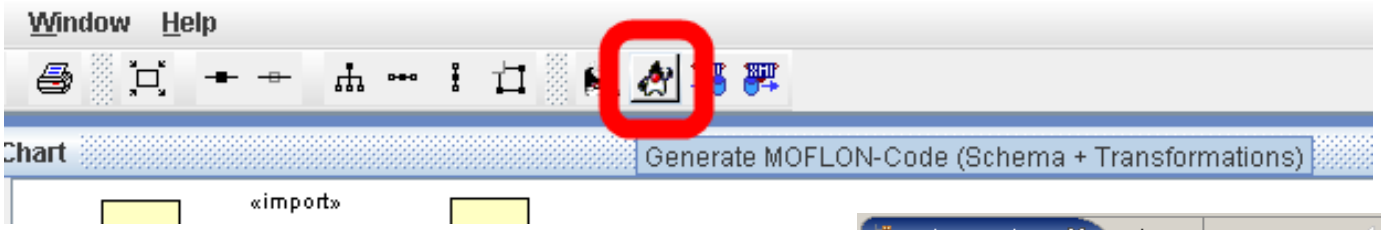


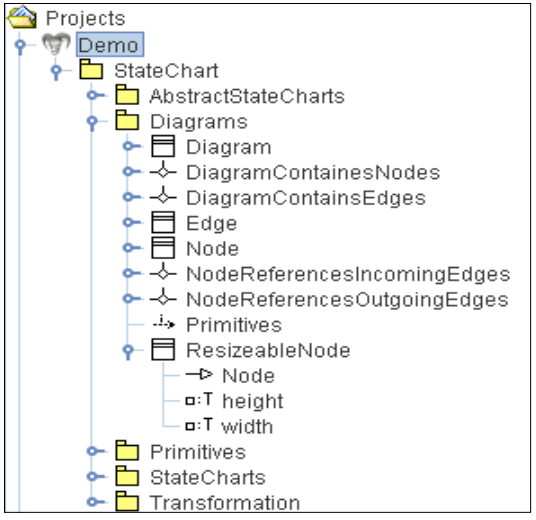
Example: 1.a) MOF Metamodel for Statecharts

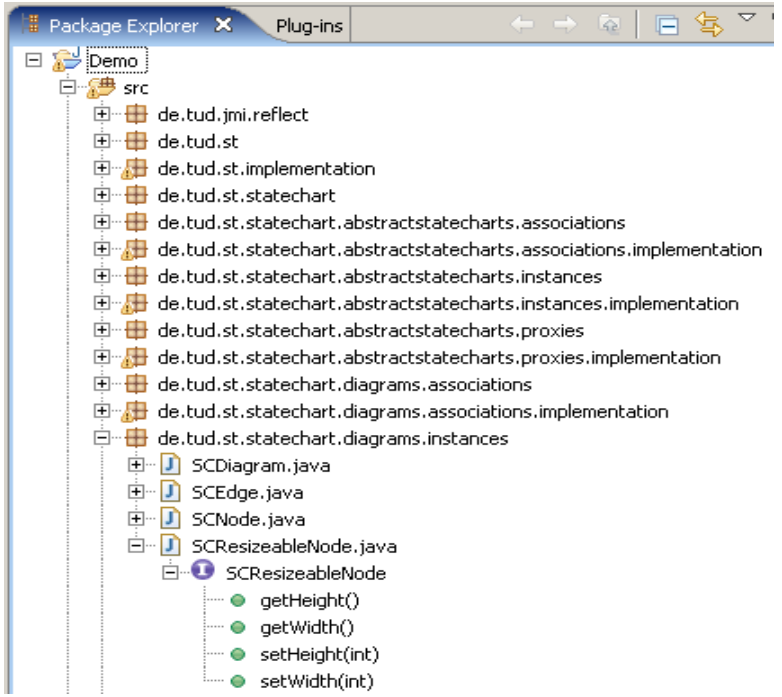


Example: 1.b) Code Generation from Statechart-Metamodel

- Uses JMI interfaces for the repository (metamodel-driven repository)
  - Codegenerator uses String template engine Velocity and XSLT-1.1 XML transformation
- Generates code for all Methods modeled as Story-diagrams (from Fujaba)



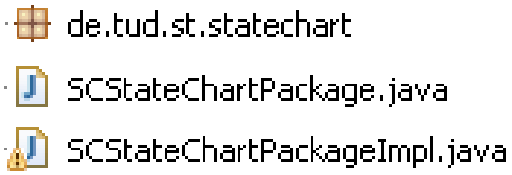




Example: 1.b) Codegeneration from Metamodel for Statecharts

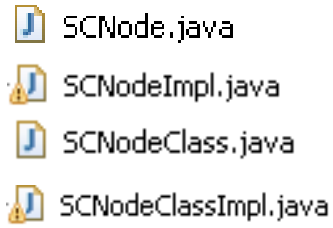
Per (E)MOF Package

- Java Package
- Interface
- Implementation



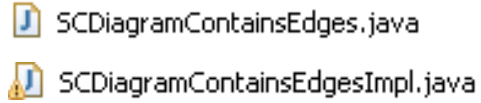
Per Metaclass

- Interface
- Implementation
- Proxy Interface
- Proxy Implementation



Per Association

- Interface
- Implementation



Example: 1.c) How to Use Statechart Models in the Generated Repository

- Initialize root package

```
SCStateChartPackage root = new SCStateChartPackageImpl();
```

- Find Proxy of repository

```
root.getSCDiagramsPackage().getSCNode();
```

- Generate nodes (model elements) via Proxy. All interfaces are typed by metaclasses

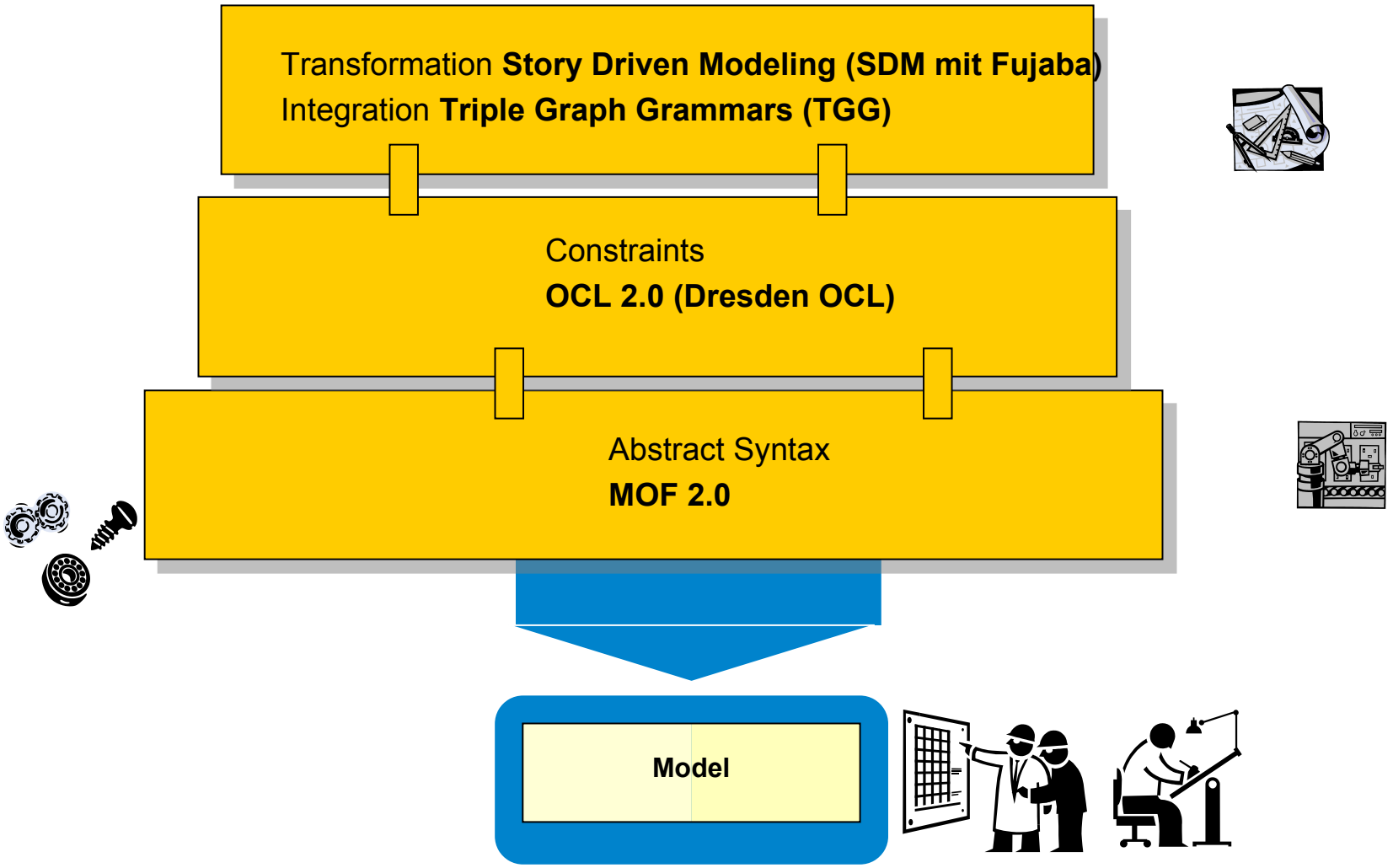
```
SCNode node = root.getSCDiagramsPackage().getSCNode().createSCNode();
```

34.2. The Metamodeling Architecture of MetaCASE Tool MOFLON

Slides from: 10 Jahre Dresden-OCL – Workshop
http://dresden-ocl.sourceforge.net/
http://dresden-ocl.sourceforge.net/10years.html
used by permission

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Metamodel Architecture of MOFLON



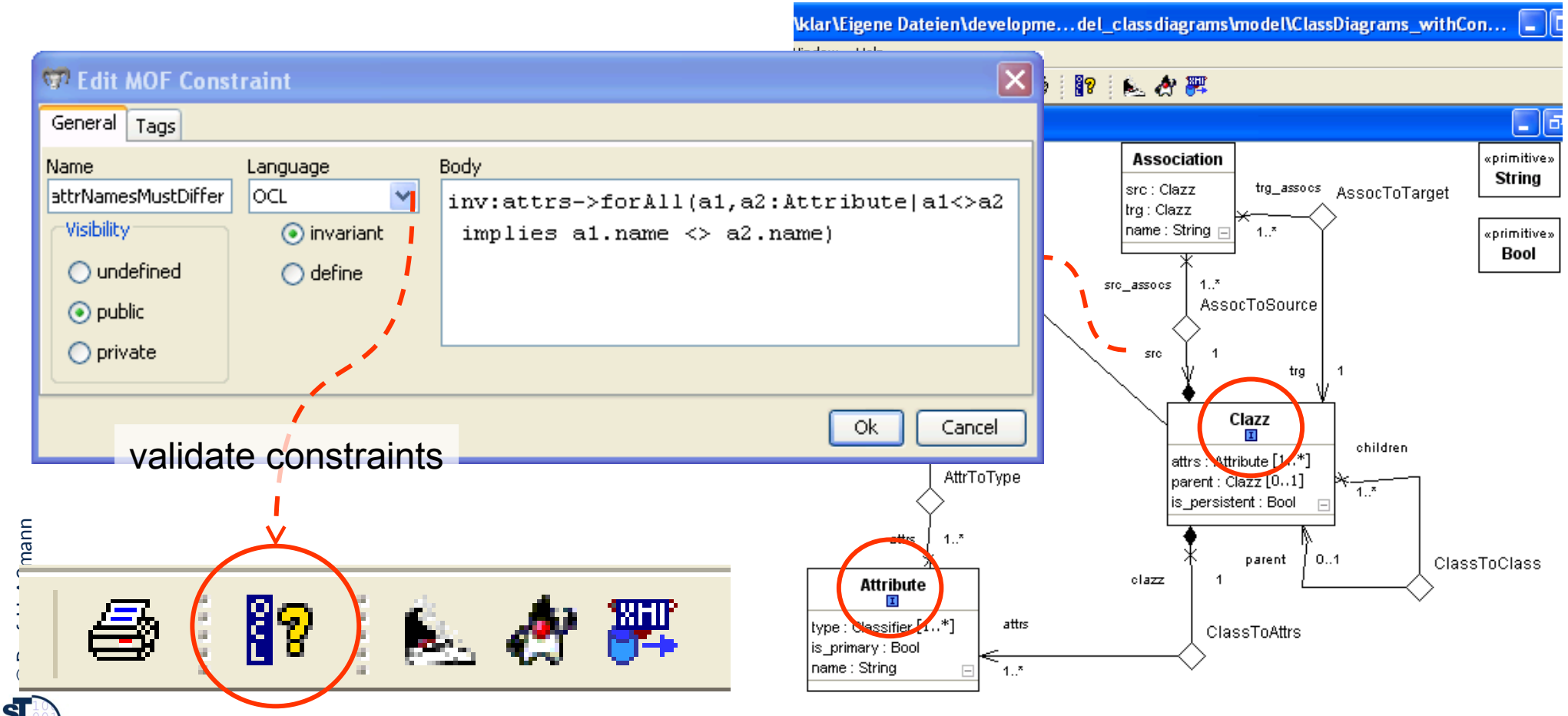
MOFLON MetaCASE – Main Features

- MOF2.0 editor (draw metamodels that comply to MOF2.0 standard)
  → build Domain Specific Languages (DSLs)
  - based on the CASE-tool framework Fujaba
  - possibility to extend MOFLON by own plugins
- interoperabilty (import / export)
- transform metamodel instances with model transformations (SDM, TGG)
- generate code (JMI-compliant) from DSLs
- instantiate models of the DSL (= repositories)
- basic editing support for generated repositories

(OCL) Constraints in MOFLON – MOF Editor

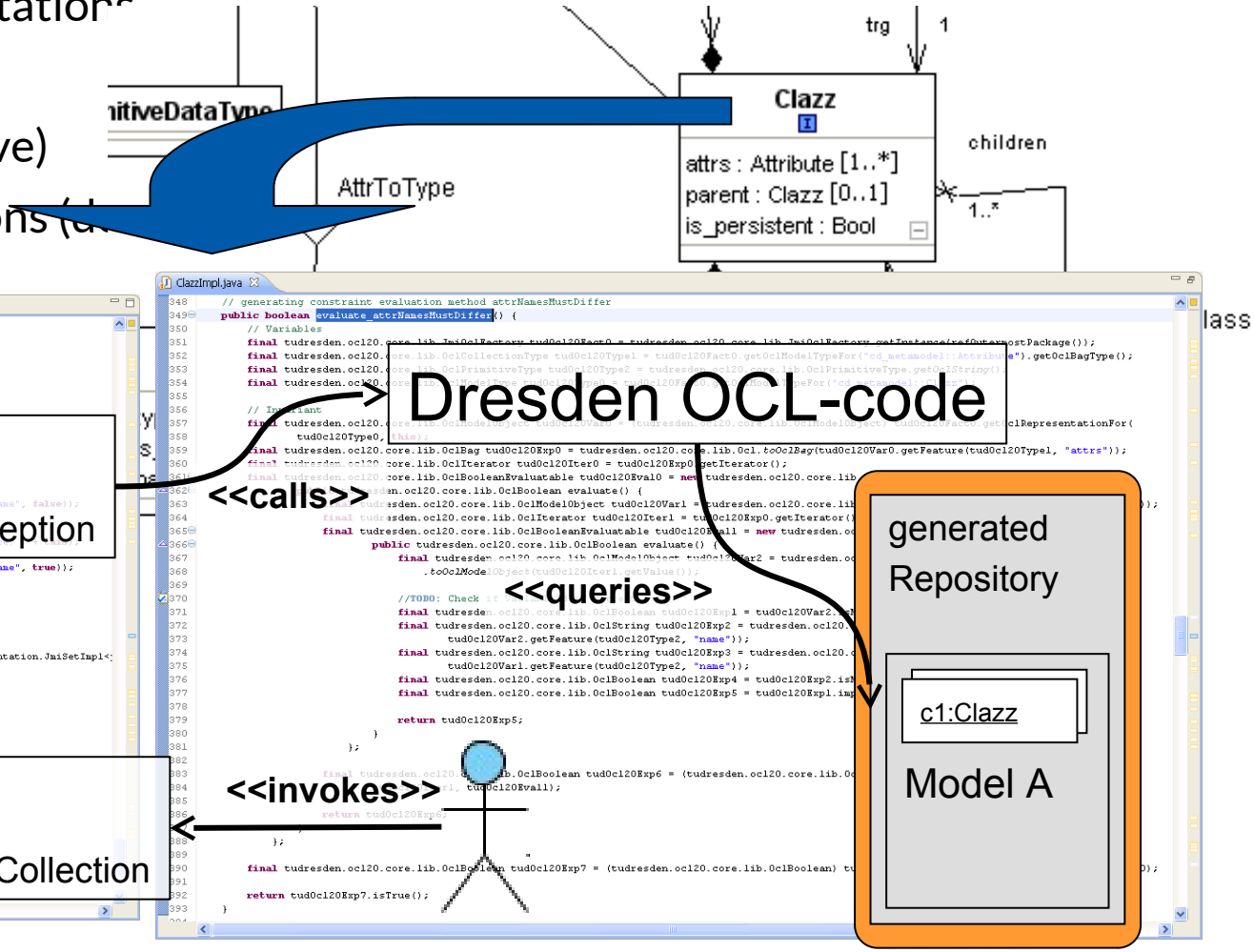
- MOF allows to add constraints to every MOF element
- MOFLON has an underlying MOF metamodel repository

→ MOFLON MOF editor may add constraints to elements



(OCL) Constraints in MOFLON – Generated Implementations

- MOFLON generates metamodel-based repositories (Java/JMI)
- MOFLON uses Dresden OCL to add constraint code to generated implementations
  - invariants (inv)
  - derived attributes (derive)
  - helper variables/functions (d



ClazzImpl.java

```
619
620    public Collection<String> refConstraintNames() {
621        Collection<String> constraintNames = new java.util.HashSet<String>();
622
623        constraintNames.add("attrNamesMustDiffer");
624
625        return constraintNames;
626    }
627
628    public javax.jmi.reflect.JmiException refVerifyConstraint(String constraintName) {
629        if ("attrNamesMustDiffer".equals(constraintName)) {
630            if (!evaluate_attrNamesMustDiffer()) {
631                String constraintBody = "unknown body";
632                constraintBody = "inv:attrs->forAll(a1,a2:Attribute|a1<>a2 implies a1.name <> a2.name)";
633                informListener(new ConstraintEvent(this, ConstraintEvent.EVENT_OCL_INVARIANT, "constraintName", false));
634
635                return new javax.jmi.reflect.ConstraintViolationException(
636                    constraintBody, this, "constraint named '" + constraintName + "' is violated in instance: " + this);
637            } else {
638                informListener(new ConstraintEvent(this, ConstraintEvent.EVENT_OCL_INVARIANT, "constraintName", true));
639            }
640        }
641        return null;
642    }
643
644    public Collection<javax.jmi.reflect.JmiException> refVerifyConstraints(boolean deepVerify) {
645        Collection<javax.jmi.reflect.JmiException> invalidConstraints = new org.moflon.collections.implementation.JmiSetImpl<
646
647        for (String constraintName : refConstraintNames()) {
648            javax.jmi.reflect.JmiException constraintException = refVerifyConstraint(constraintName);
649
650            if (constraintException != null) {
651                invalidConstraints.add(constraintException);
652            }
653        }
654
655        if (deepVerify) {
656        }
657
658        if (invalidConstraints.size() > 0) {
659            return invalidConstraints;
660        } else {
661            return null;
662        }
663    }
664
```

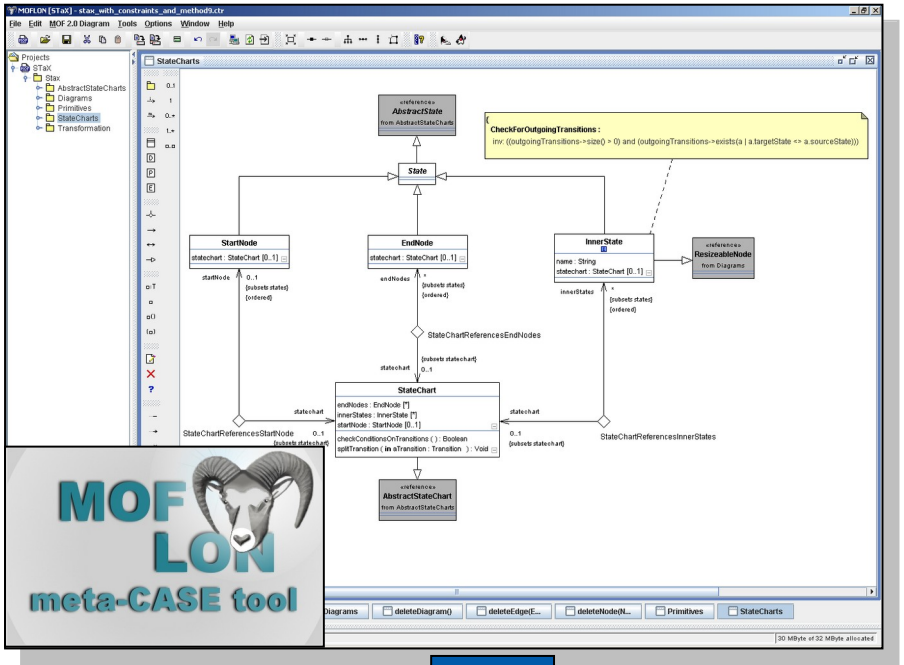
JMI compiliant method

ClazzImpl.java

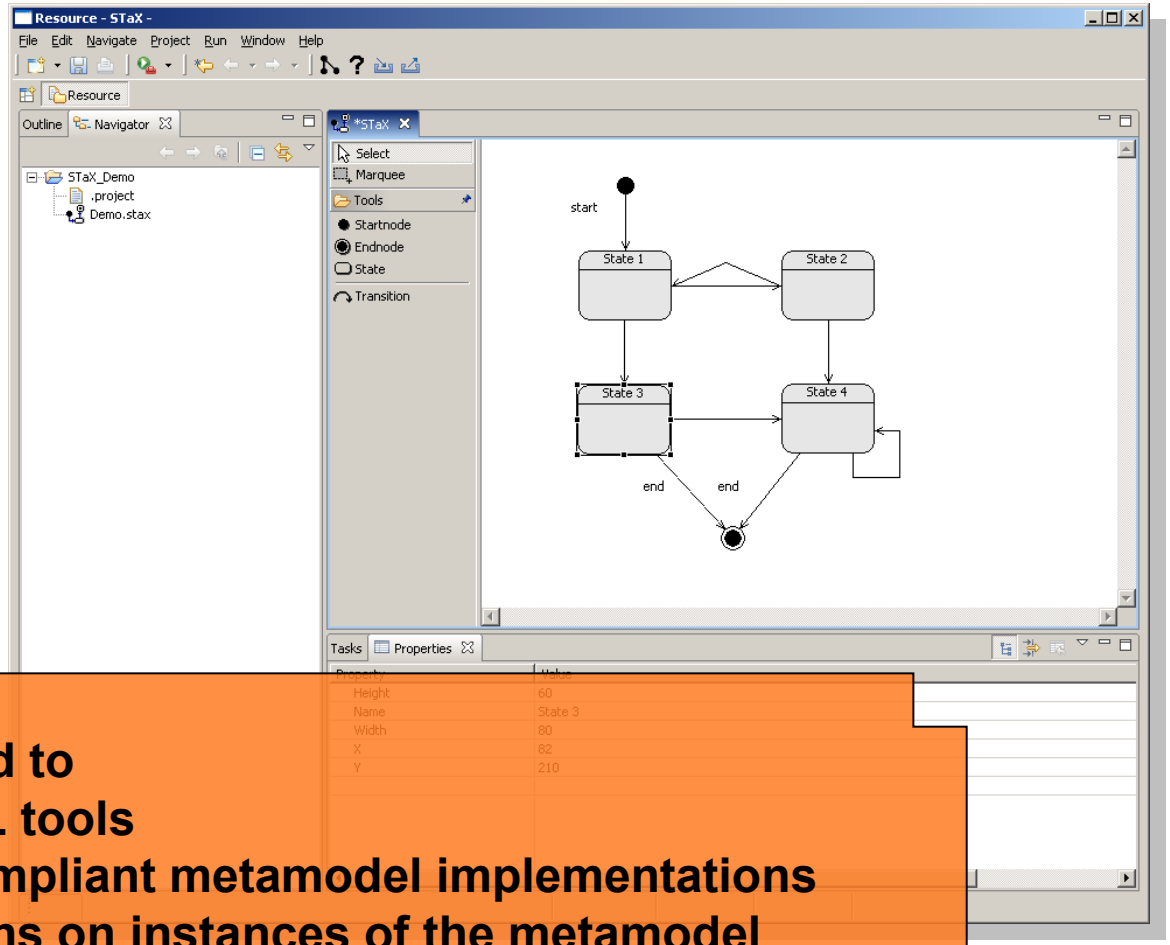
```
348    // generating constraint evaluation method attrNamesMustDiffer
349    public boolean evaluate_attrNamesMustDiffer() {
350        // Variables
351        final tudresden.ocl20.core.lib.JmiOclFactory tudOcl20Fact0 = tudresden.ocl20.core.lib.JmiOclFactory.getInstance(refOutermostPackage());
352        final tudresden.ocl20.core.lib.OclCollectionType tudOcl20Type1 = tudOcl20Fact0.getOclModelTypeFor("cd_metamodel::Attribute").getOclBagType();
353        final tudresden.ocl20.core.lib.OclPrimitiveType tudOcl20Type2 = tudresden.ocl20.core.lib.OclPrimitiveType.getOclString();
354        final tudresden.ocl20.core.lib.OclModelType tudOcl20Type0 = tudOcl20Fact0.getOclModelTypeFor("cd_metamodel::Clazz");
355
356        // Invariant
357        final tudresden.ocl20.core.lib.OclModelObject tudOcl20Var0 = (tudresden.ocl20.core.lib.OclModelObject) tudOcl20Fact0.getOclRepresentationFor(
358                tudOcl20Type0, this);
359        final tudresden.ocl20.core.lib.OclBag tudOcl20Exp0 = tudresden.ocl20.core.lib.Ocl.toOclBag(tudOcl20Var0.getFeature(tudOcl20Type1, "attrs"));
360        final tudresden.ocl20.core.lib.OclIterator tudOcl20Iter0 = tudOcl20Exp0.getIterator();
361        final tudresden.ocl20.core.lib.OclBooleanEvaluatable tudOcl20Eval0 = new tudresden.ocl20.core.lib.OclBooleanEvaluatable() {
362            public tudresden.ocl20.core.lib.OclBoolean evaluate() {
363                final tudresden.ocl20.core.lib.OclModelObject tudOcl20Var1 = tudresden.ocl20.core.lib.Ocl.toOclModelObject(tudOcl20Iter0.getValue());
364                final tudresden.ocl20.core.lib.OclIterator tudOcl20Iter1 = tudOcl20Exp0.getIterator();
365                final tudresden.ocl20.core.lib.OclBooleanEvaluatable tudOcl20Eval1 = new tudresden.ocl20.core.lib.OclBooleanEvaluatable() {
366                    public tudresden.ocl20.core.lib.OclBoolean evaluate() {
367                        final tudresden.ocl20.core.lib.OclModelObject tudOcl20Var2 = tudresden.ocl20.core.lib.Ocl
368                            .toOclModelObject(tudOcl20Iter1.getValue());
369
370                        //TODO: Check if VariableId is correct
371                        final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp1 = tudOcl20Var2.isNotEqualTo(tudOcl20Var1);
372                        final tudresden.ocl20.core.lib.OclString tudOcl20Exp2 = tudresden.ocl20.core.lib.Ocl.toOclString(
373                                tudOcl20Var2.getFeature(tudOcl20Type2, "name"));
374                        final tudresden.ocl20.core.lib.OclString tudOcl20Exp3 = tudresden.ocl20.core.lib.Ocl.toOclString(
375                                tudOcl20Var1.getFeature(tudOcl20Type2, "name"));
376                        final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp4 = tudOcl20Exp2.isNotEqualTo(tudOcl20Exp3);
377                        final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp5 = tudOcl20Exp1.implies(tudOcl20Exp4);
378
379                        return tudOcl20Exp5;
380                    }
381                };
382
383                final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp6 = (tudresden.ocl20.core.lib.OclBoolean) tudOcl20Exp0.forAll(
384                        tudOcl20Iter1, tudOcl20Eval1);
385
386                return tudOcl20Exp6;
387            }
388        };
389
390        final tudresden.ocl20.core.lib.OclBoolean tudOcl20Exp7 = (tudresden.ocl20.core.lib.OclBoolean) tudOcl20Exp0.forAll(tudOcl20Iter0, tudOcl20Eval0);
391
392        return tudOcl20Exp7.isTrue();
393    }
```

Generated Code from Dresden OCL

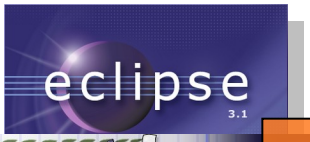
Result of MOFLON Example 1 – Statechart Editor (STaX)



Editor:
- data structure (MOFLON repository)
- GUI (GEF)

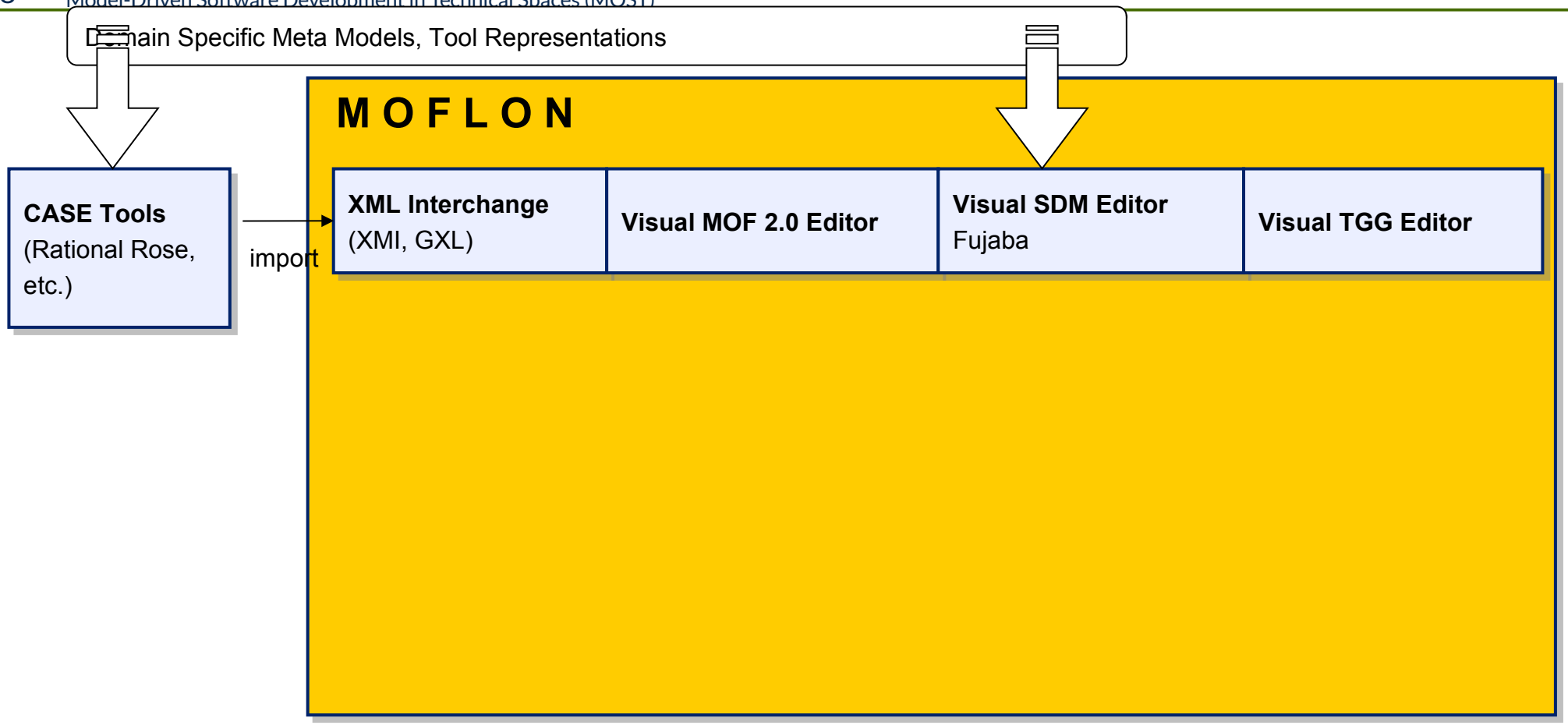


+

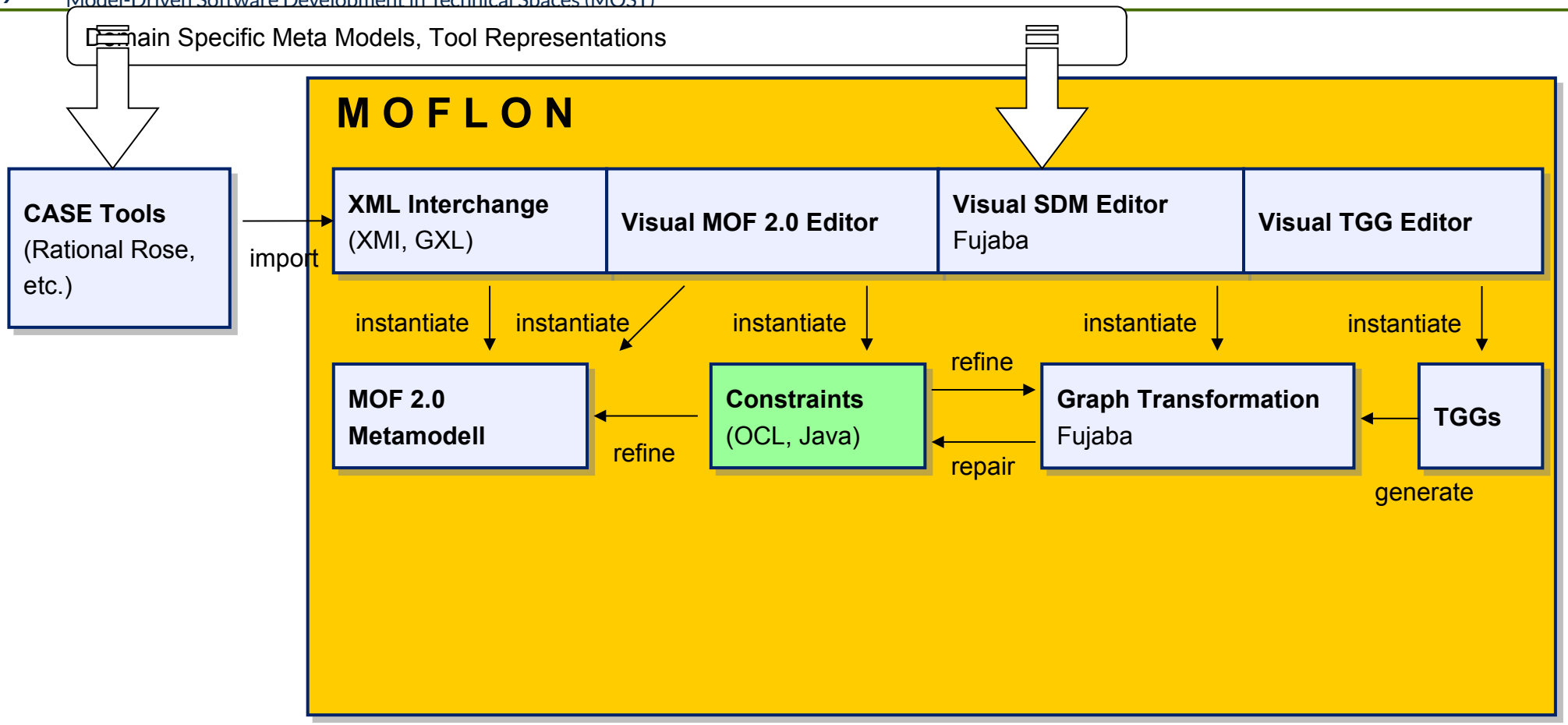


MOFLON is mainly used to
- **integrate existing DSL tools**
- **generate standard compliant metamodel implementations**
- **specify transformations on instances of the metamodel**

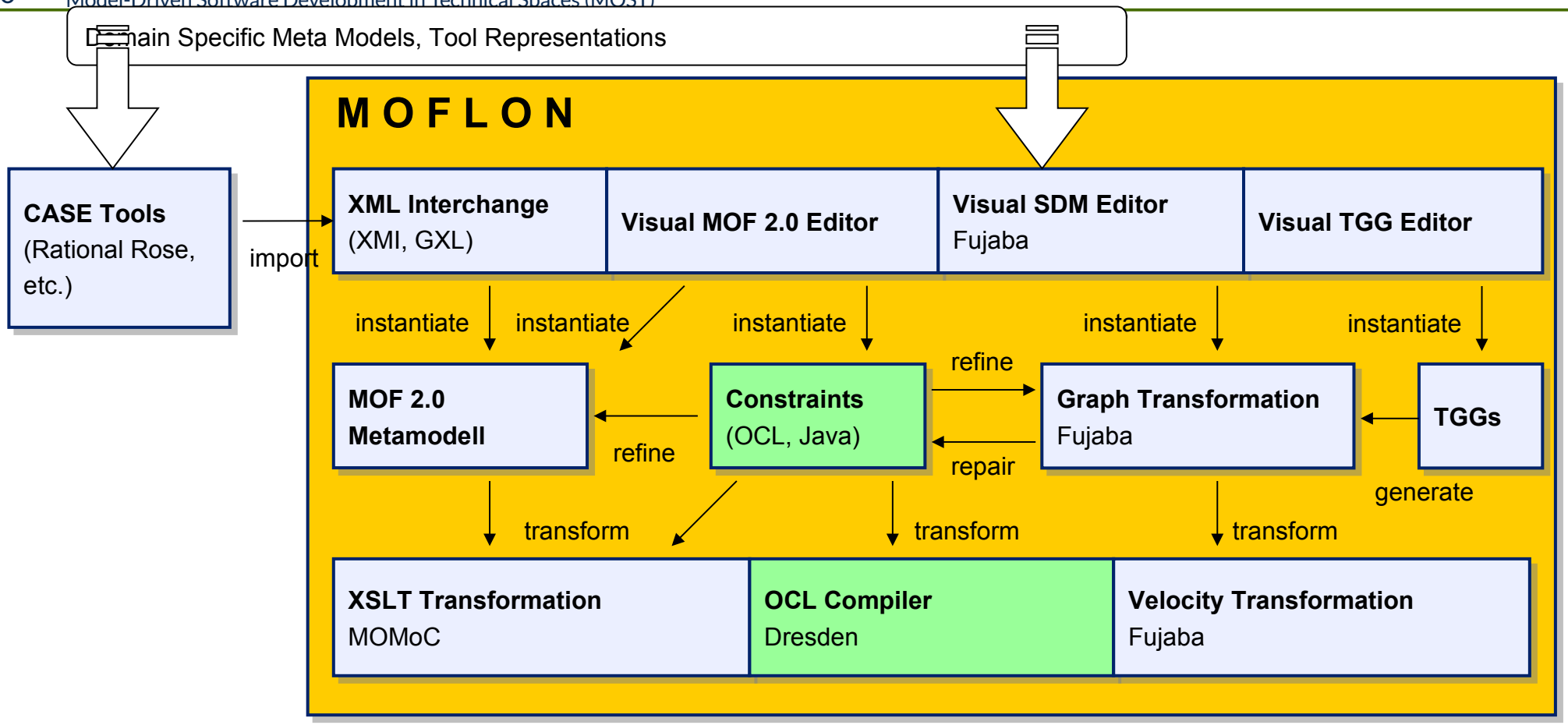
34.3.3 MOFLON – Architecture



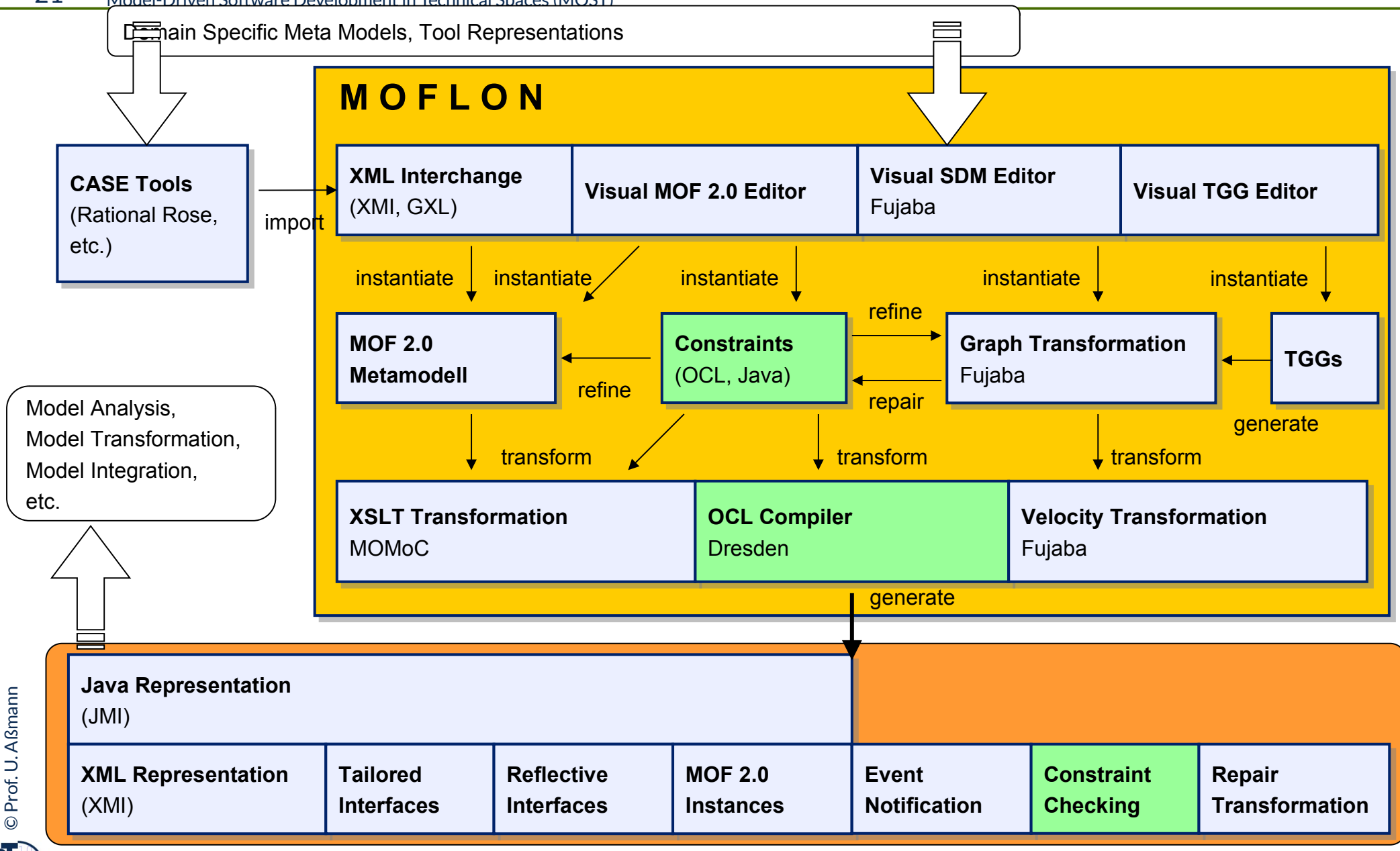
MOFLON – Architecture



MOFLON – Architecture



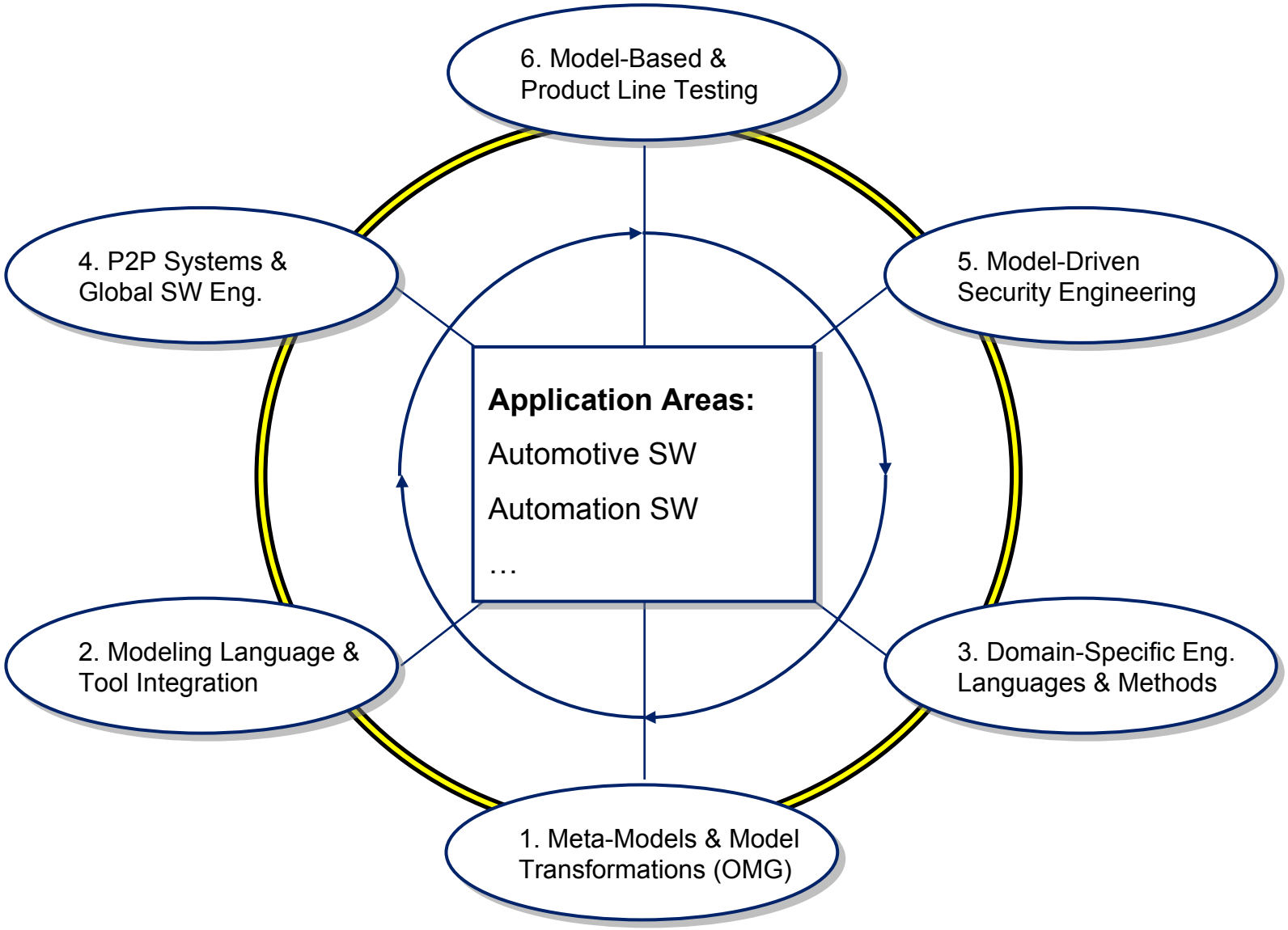
MOFLON – Architecture



MOFLON is Bootstrapped

- TU Darmstadt bootstraps the MOFLON MOF Metamodel periodically
  - Since 2013, ported to EMOF (eMOFLON)
- Bootstrap has important advantages:
  - If more OCL constraints are added to the (e)MOF metamodel
  - Regenerate MOFLON MOF implementation
  - Activate the extended constraint checking in MOFLON (model verification, model consistency checking, model wellformedness)

Model-Driven Software Development at Real-Time Systems Lab (Prof. Schürr)



Related Approaches

standards | approaches based on graph-/modeltransformation | classic meta-CASE approaches | text based approaches

	MOF, OCL, QVT	MOFLON	Fujaba & TGG	Progres & TGG	GME & GReAT	EMF & Tefkat	AToM³	MetaEdit+	Microsoft DSL	EMF & GMF	Pounamu	DiaGen	EBNF & TXL	SQL	XML
Abstract syntax	+	+	+	+	o	o	o	+	+	o	+	+	+	o	+
Concrete syntax	--	--	--	+	+	--	+	+	+	+	+	+	--	--	--
Static semantics	+	+	o	+	+	+	o	o	--	+	o	+	o	o	--
Dynamic semantics	+	+	+	+	+	+	+	o	o	--	--	--	+	--	o
Model analysis	+	+	+	+	+	o	+	o	--	+	--	o	+	o	+
Model transformation	+	+	+	+	+	+	+	o	--	--	--	o	+	o	+
Model integration	+	+	+	+	o	+	--	--	--	--	--	--	o	--	o
Acceptability	+	+	o	--	o	+	--	+	--	o	+	o	o	+	+
Scaleability	+	+	--	o	--	o	--	o	--	--	--	--	--	--	o
Tool availability	--	o	o	+	+	+	+	+	o	o	+	+	+	+	o
Expressiveness	+	+	o	+	+	o	o	o	o	o	o	o	+	o	o

from Amelunxen, Königs, Rötschke, and Schürr,
„MOSL: Composing a Visual Language for a Metamodeling Framework“
in IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC 2006), September, 2006, 81-84

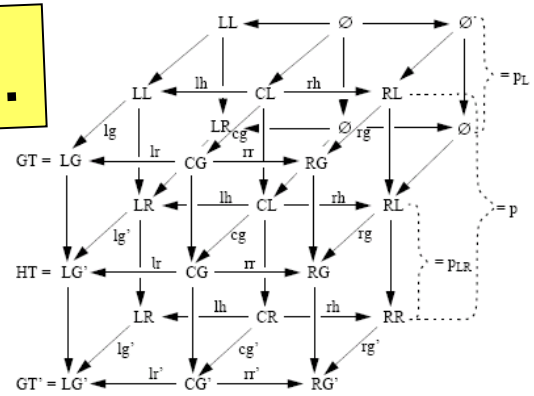
Further reading

- A. Königs, A. Schürr: "Tool Integration with Triple Graph Grammars - A Survey", in: R. Heckel (ed.), Proceedings of the SegraVis School on Foundations of Visual Modelling Techniques, Amsterdam: Elsevier Science Publ., 2006; Electronic Notes in Theoretical Computer Science, Vol. 148, 113-150.
- F. Klar, S. Rose, A. Schürr: "TiE - A Tool Integration Environment", Proceedings of the 5th ECMDA Traceability Workshop, 2009; CTIT Workshop Proceedings, Vol. WP09-09, 39-48
- F. Klar, S. Rose, A. Schürr: "A Meta-Model-Driven Tool Integration Development Process", Proceedings of the 2nd International United Information Systems Conference, 2008; Lecture Notes in Business Information Processing, 201-212.
- C. Amelunxen, A. Königs, T. Rötschke, A. Schürr: "MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations", in: A. Rensink, J. Warmer (eds.), Model Driven Architecture - Foundations and Applications: Second European Conference, Heidelberg: Springer Verlag, 2006; Lecture Notes in Computer Science (LNCS), Vol. 4066, Springer Verlag, 361-375.
- A. Königs: "Model Integration and Transformation - A Triple Graph Grammar-based QVT Implementation", Technische Universität Darmstadt, Phd Thesis, 2009.

The End

Some slides are courtesy Florian Heidenreich and Felix Klar

Thank you for your attention…



http://www.emoflon.org