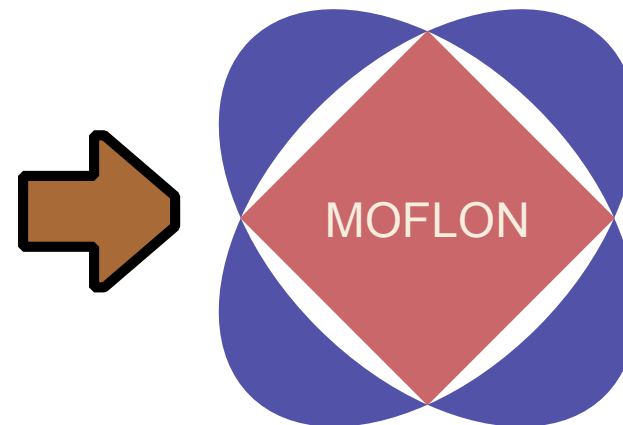


34. The Meta-CASE-Tool (e)MOFLON

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



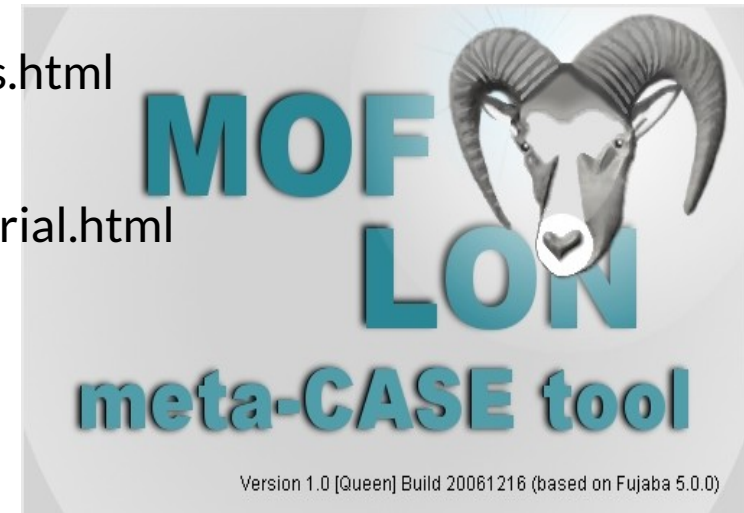
DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

Reading

2

Model-Driven Software Development in Technical Spaces (MOST)

- ▶ 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>



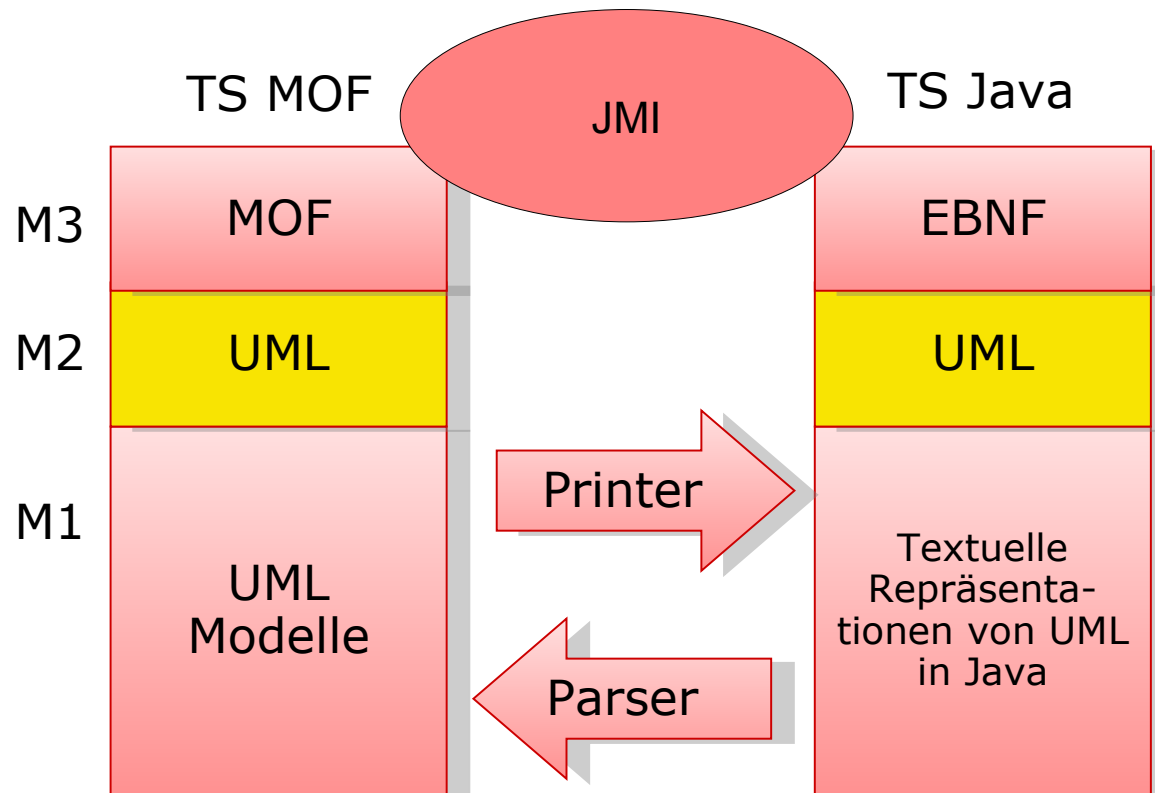
34.1. eMOFLON Introduction

- ▶ 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)

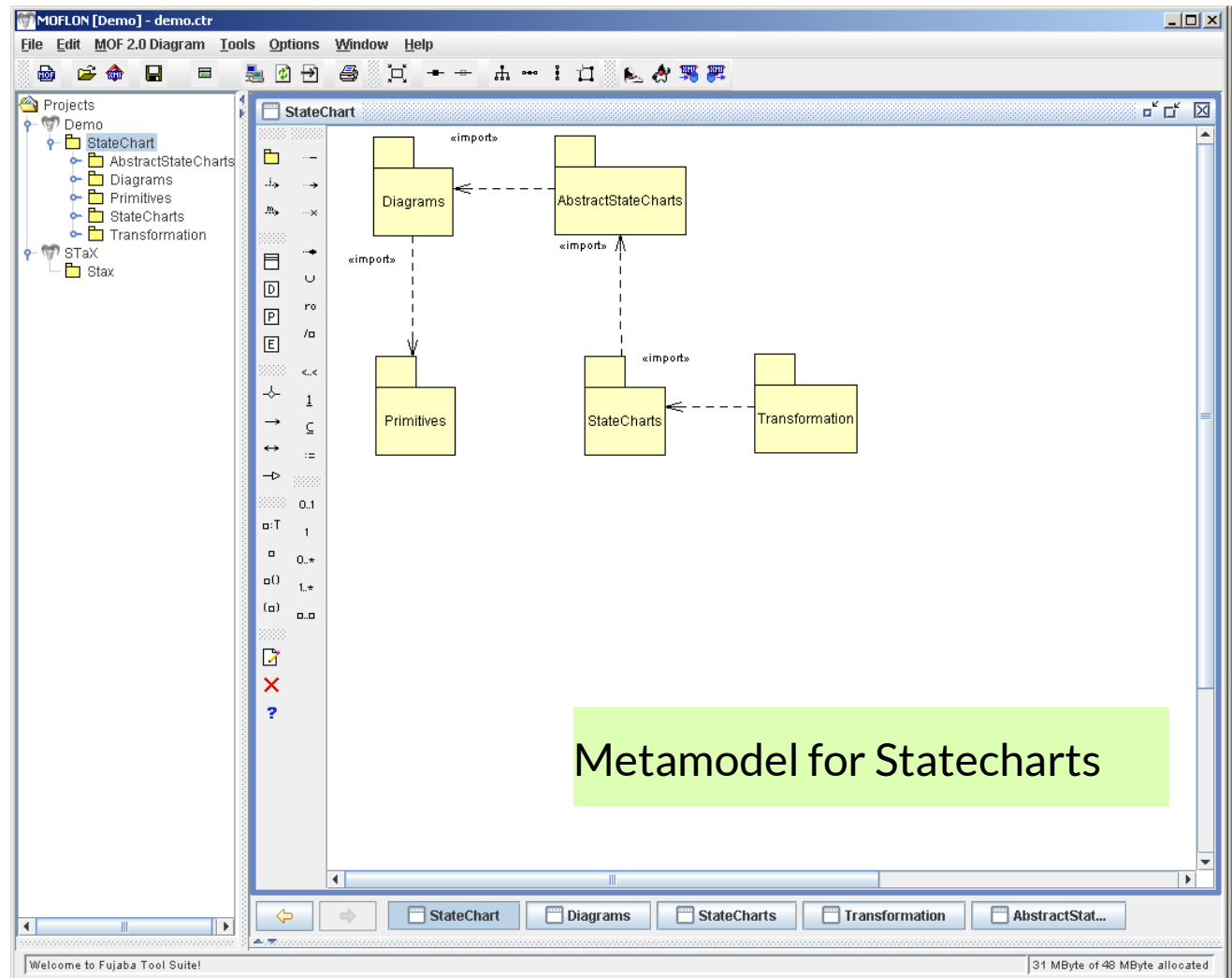


(e)MOFLON Example 1: Metamodel for Statecharts: Development Process

5

Model-Driven Software Development in Technical Spaces (MOST)

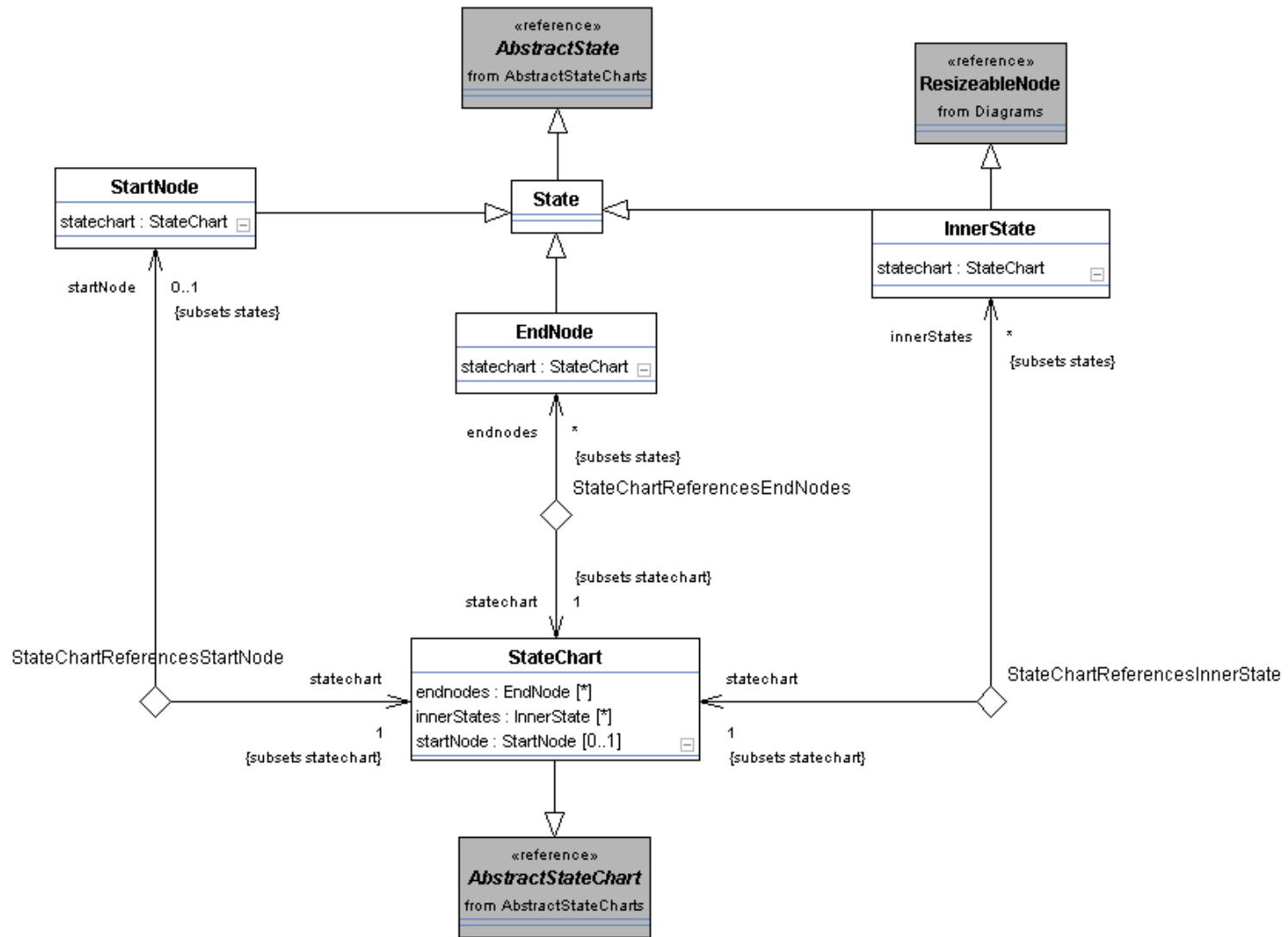
- 1) Create metamodel
- 2) Generate Code (Generate repository with constraint-checker) with the JMI interfaces



Example: 1.a) MOF Metamodel for Statecharts

6

Model-Driven Software Development in Technical Spaces (MOST)

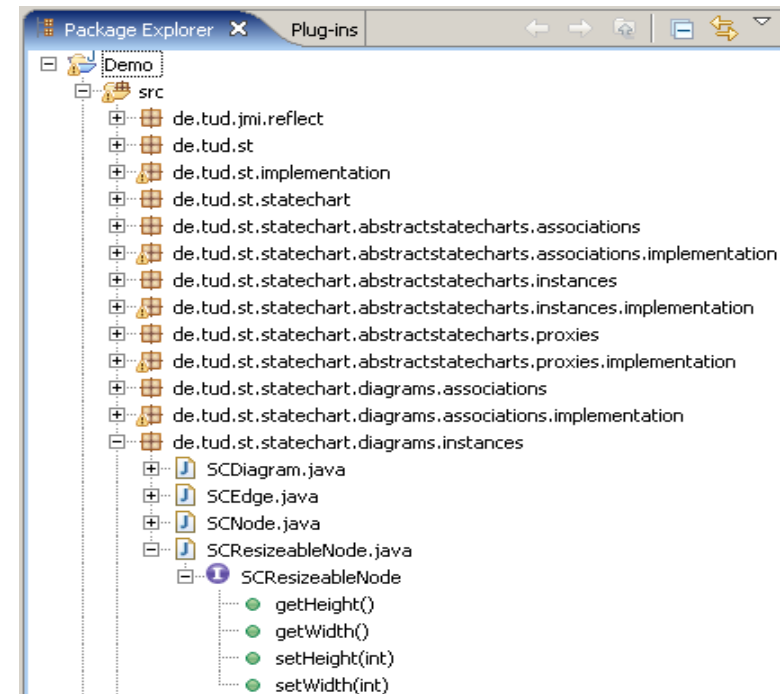
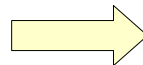
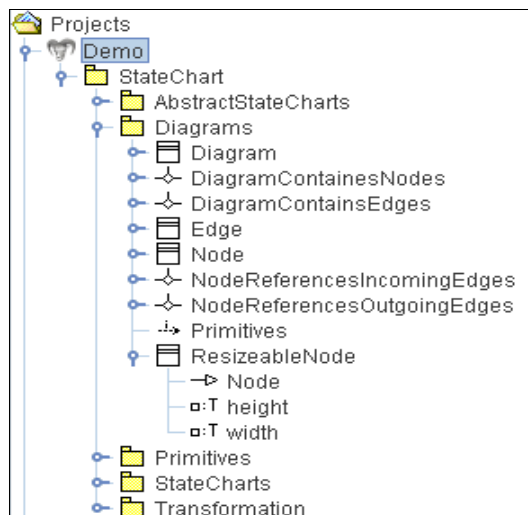
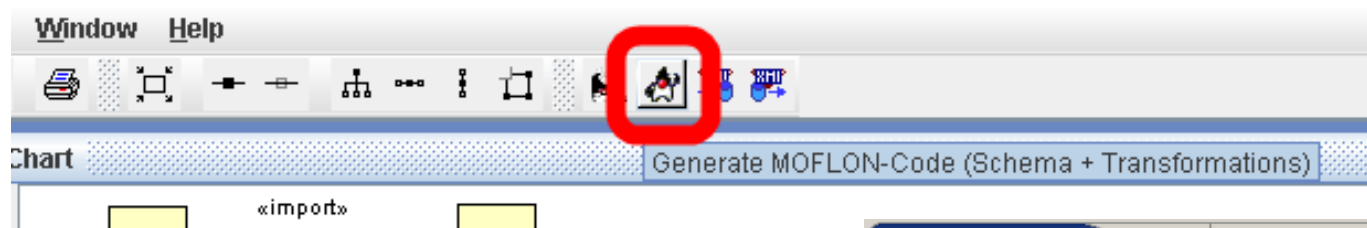


Example: 1.b) Code Generation from Statechart-Metamodel

7




Model-Driven Software Development in Technical Spaces (MOST)

- ▶ 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  de.tud.st.statechart
- Interface  SCStateChartPackage.java
- Implementation  SCStateChartPackageImpl.java

Per Metaclass

- Interface  SCNode.java
- Implementation  SCNodeImpl.java
- Proxy Interface  SCNodeClass.java
- Proxy Implementation  SCNodeClassImpl.java

Per Association

- Interface  SCDiagramContainsEdges.java
- Implementation  SCDiagramContainsEdgesImpl.java

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 ();
```

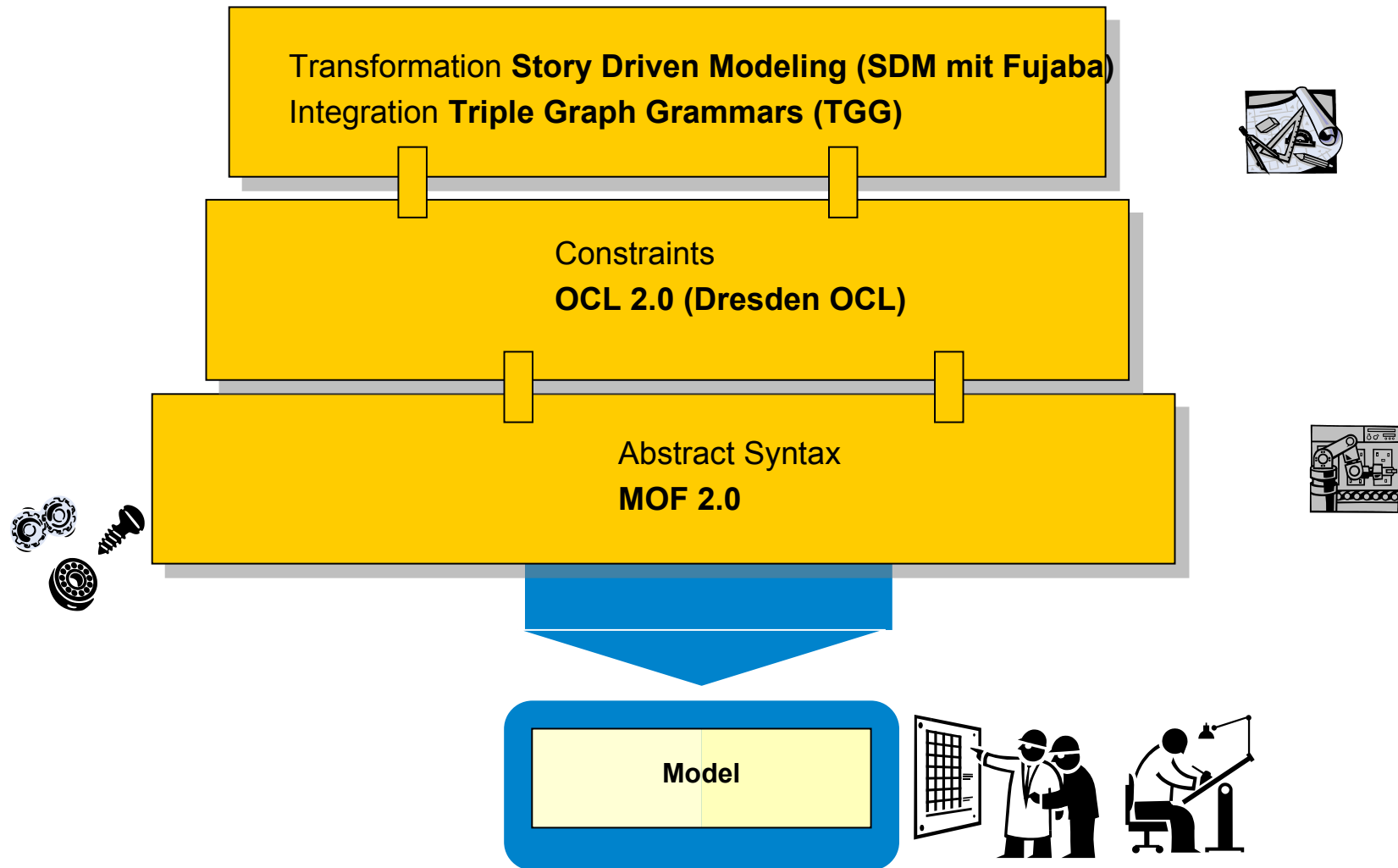
34.2. The Metamodeling Architecture of MetaCASE Tool MOFLON

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

12

Model-Driven Software Development in Technical Spaces (MOST)

- ▶ 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
- ▶ interoperability (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

13

Model-Driven Software Development in Technical Spaces (MOST)

- ▶ MOF allows to add constraints to every MOF element
 - ▶ MOFLON has an underlying MOF metamodel repository
- MOFLON MOF editor may add constraints to elements

The screenshot displays the MOFLON MOF Editor interface. A dialog box titled "Edit MOF Constraint" is open, showing the "General" tab. The "Name" field contains "attrNamesMustDiffer", the "Language" is set to "OCL", and the "Body" contains the OCL expression: `inv:attrs->forAll(a1,a2:Attribute|a1<>a2 implies a1.name <> a2.name)`. The "invariant" radio button is selected. The "Visibility" section shows "public" selected. A red dashed arrow points from the "validate constraints" text to the question mark icon in the toolbar. The background shows a class diagram with classes "Association", "Clazz", and "Attribute". The "Clazz" class is circled in red, and the "Attribute" class is also circled in red. The diagram shows relationships like "AssocToSource", "AssocToTarget", "AttrToType", "ClassToAttrs", and "ClassToClass".

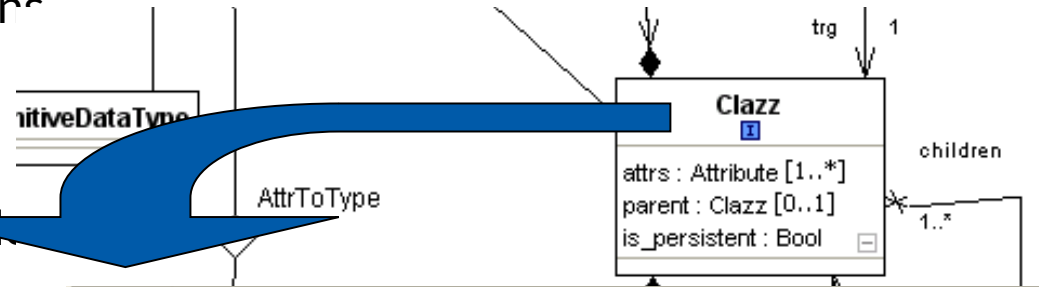
validate constraints

mann



(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 (def)



```

619 public Collection<String> refConstraintNames() {
620     Collection<String> constraintNames = new java.util.HashSet<String>();
621     ...
622     constraintNames.add("attrNamesMustDiffer");
623     ...
624     return constraintNames;
625 }
626
627 public boolean refVerifyConstraint(String constraintName)
628     throws JmiException {
629     ...
630     return true;
631 }
632
633 public Collection<JavaxJmiReflectJmiException> refVerifyConstraints(boolean deepVerify) {
634     Collection<JavaxJmiReflectJmiException> invalidConstraints = new ArrayList<JavaxJmiReflectJmiException>();
635     for (String constraintName : refConstraintNames()) {
636         javax.jmi.reflect.JmiException constraintException = refVerifyConstraint(constraintName);
637         if (constraintException != null) {
638             invalidConstraints.add(constraintException);
639         }
640     }
641     return invalidConstraints;
642 }
643
644 public Collection<JavaxJmiReflectJmiException> refVerifyConstraints(boolean deepVerify) {
645     Collection<JavaxJmiReflectJmiException> invalidConstraints = new ArrayList<JavaxJmiReflectJmiException>();
646     for (String constraintName : refConstraintNames()) {
647         javax.jmi.reflect.JmiException constraintException = refVerifyConstraint(constraintName);
648         if (constraintException != null) {
649             invalidConstraints.add(constraintException);
650         }
651     }
652     return invalidConstraints;
653 }
        
```

MOFLON-code
refVerifyConstraint(String name):JmiException

JMI compliant method
refVerifyConstraints(boolean deepVerify):Collection

```

949 // generating constraint evaluation method attrNamesMustDiffer
950 public boolean Evaluate_attrNamesMustDiffer() {
951     // Variables
952     final tudresden.oc120.core.lib.OclFactory tud0cl20Fact0 = tudresden.oc120.core.lib.OclFactory.getFactoryForPackage();
953     final tudresden.oc120.core.lib.OclCollectionType tud0cl20Type1 = tud0cl20Fact0.getOclModelTypeFor("cd_watamodel::Attribute");
954     final tudresden.oc120.core.lib.OclPrimitiveType tud0cl20Type2 = tudresden.oc120.core.lib.OclPrimitiveType.getOclStringType();
955     final tudresden.oc120.core.lib.OclBoolean tud0cl20Bool = tudresden.oc120.core.lib.OclBoolean.getOclBooleanType();
956     // Tracing
957     final tudresden.oc120.core.lib.OclModelObject tud0cl20Var1 = tudresden.oc120.core.lib.OclModelObject.getOclRepresentationFor(
958         tud0cl20Type1, null);
959     final tudresden.oc120.core.lib.OclBoolean tud0cl20Exp0 = tudresden.oc120.core.lib.OclBoolean.getOclBooleanType().getFeature(tud0cl20Type1, "attrs");
960     final tudresden.oc120.core.lib.OclIterator tud0cl20Iter0 = tud0cl20Exp0.getIterator();
961     final tudresden.oc120.core.lib.OclBooleanEvaluatable tud0cl20Eval0 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
962         public boolean evaluate() {
963             final tudresden.oc120.core.lib.OclModelObject tud0cl20Var1 = tudresden.oc120.core.lib.OclModelObject.getOclRepresentationFor(
964                 tud0cl20Type1, null);
965             final tudresden.oc120.core.lib.OclIterator tud0cl20Iter1 = tud0cl20Exp0.getIterator();
966             final tudresden.oc120.core.lib.OclBooleanEvaluatable tud0cl20Eval1 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
967                 public boolean evaluate() {
968                     final tudresden.oc120.core.lib.OclModelObject tud0cl20Var2 = tudresden.oc120.core.lib.OclModelObject.getOclRepresentationFor(
969                         tud0cl20Type1, null);
970                     final tudresden.oc120.core.lib.OclString tud0cl20Exp1 = tudresden.oc120.core.lib.OclString.getOclStringType().getFeature(tud0cl20Type2, "name");
971                     final tudresden.oc120.core.lib.OclString tud0cl20Exp2 = tudresden.oc120.core.lib.OclString.getOclStringType().getFeature(tud0cl20Type2, "name");
972                     final tudresden.oc120.core.lib.OclBoolean tud0cl20Exp3 = tudresden.oc120.core.lib.OclBoolean.getOclBooleanType().getFeature(tud0cl20Type2, "name");
973                     final tudresden.oc120.core.lib.OclBoolean tud0cl20Exp4 = tud0cl20Exp2.isEqual(tud0cl20Exp1);
974                     final tudresden.oc120.core.lib.OclBoolean tud0cl20Exp5 = tud0cl20Exp3.isTrue();
975                     return tud0cl20Exp4 && tud0cl20Exp5;
976                 }
977             };
978             return tud0cl20Iter1.evaluate(tud0cl20Eval1);
979         }
980     };
981     return tud0cl20Eval0.evaluate();
982 }
983
984 final tudresden.oc120.core.lib.OclBoolean tud0cl20Exp6 = (tudresden.oc120.core.lib.OclBoolean) tud0cl20Eval1.evaluate();
985 return tud0cl20Exp6.isTrue();
986 }
987
988 final tudresden.oc120.core.lib.OclBoolean tud0cl20Exp7 = (tudresden.oc120.core.lib.OclBoolean) tud0cl20Eval1.evaluate();
989 return tud0cl20Exp7.isTrue();
990 }
991
992 return tud0cl20Exp7.isTrue();
993 }
        
```

Dresden OCL-code

<<calls>>

<<queries>>

<<invokes>>

generated Repository

c1:Clazz

Model A



```
ClazzImpl.java X
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 compliant
method

```
ClazzImpl.java X
348 // generating constraint evaluation method attrNamesMustDiffer
349 public boolean evaluate_attrNamesMustDiffer() {
350     // Variables
351     final tudresden.oc120.core.lib.JmiOclFactory tudOcl20Fact0 = tudresden.oc120.core.lib.JmiOclFactory.getInstance(refOutermostPackage());
352     final tudresden.oc120.core.lib.OclCollectionType tudOcl20Type1 = tudOcl20Fact0.getOclModelTypeFor("cd_metamodel::Attribute").getOclBagType();
353     final tudresden.oc120.core.lib.OclPrimitiveType tudOcl20Type2 = tudresden.oc120.core.lib.OclPrimitiveType.getOclString();
354     final tudresden.oc120.core.lib.OclModelType tudOcl20Type0 = tudOcl20Fact0.getOclModelTypeFor("cd_metamodel::Clazz");
355
356     // Invariant
357     final tudresden.oc120.core.lib.OclModelObject tudOcl20Var0 = (tudresden.oc120.core.lib.OclModelObject) tudOcl20Fact0.getOclRepresentationFor(
358         tudOcl20Type0, this);
359     final tudresden.oc120.core.lib.OclBag tudOcl20Exp0 = tudresden.oc120.core.lib.Ocl.toOclBag(tudOcl20Var0.getFeature(tudOcl20Type1, "attrs"));
360     final tudresden.oc120.core.lib.OclIterator tudOcl20Iter0 = tudOcl20Exp0.getIterator();
361     final tudresden.oc120.core.lib.OclBooleanEvaluatable tudOcl20Eval0 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
362         public tudresden.oc120.core.lib.OclBoolean evaluate() {
363             final tudresden.oc120.core.lib.OclModelObject tudOcl20Var1 = tudresden.oc120.core.lib.Ocl.toOclModelObject(tudOcl20Iter0.getValue());
364             final tudresden.oc120.core.lib.OclIterator tudOcl20Iter1 = tudOcl20Exp0.getIterator();
365             final tudresden.oc120.core.lib.OclBooleanEvaluatable tudOcl20Eval1 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
366                 public tudresden.oc120.core.lib.OclBoolean evaluate() {
367                     final tudresden.oc120.core.lib.OclModelObject tudOcl20Var2 = tudresden.oc120.core.lib.Ocl
368                         .toOclModelObject(tudOcl20Iter1.getValue());
369
370                     //TODO: Check if VariableId is correct
371                     final tudresden.oc120.core.lib.OclBoolean tudOcl20Exp1 = tudOcl20Var2.isNotEqualTo(tudOcl20Var1);
372                     final tudresden.oc120.core.lib.OclString tudOcl20Exp2 = tudresden.oc120.core.lib.Ocl.toOclString(
373                         tudOcl20Var2.getFeature(tudOcl20Type2, "name"));
374                     final tudresden.oc120.core.lib.OclString tudOcl20Exp3 = tudresden.oc120.core.lib.Ocl.toOclString(
375                         tudOcl20Var1.getFeature(tudOcl20Type2, "name"));
376                     final tudresden.oc120.core.lib.OclBoolean tudOcl20Exp4 = tudOcl20Exp2.isNotEqualTo(tudOcl20Exp3);
377                     final tudresden.oc120.core.lib.OclBoolean tudOcl20Exp5 = tudOcl20Exp1.implies(tudOcl20Exp4);
378
379                     return tudOcl20Exp5;
380                 }
381             };
382
383             final tudresden.oc120.core.lib.OclBoolean tudOcl20Exp6 = (tudresden.oc120.core.lib.OclBoolean) tudOcl20Exp0.forAll(
384                 tudOcl20Iter1, tudOcl20Eval1);
385
386             return tudOcl20Exp6;
387         }
388     };
389
390     final tudresden.oc120.core.lib.OclBoolean tudOcl20Exp7 = (tudresden.oc120.core.lib.OclBoolean) tudOcl20Exp0.forAll(tudOcl20Iter0, tudOcl20Eval0);
391
392     return tudOcl20Exp7.isTrue();
393 }
394
```

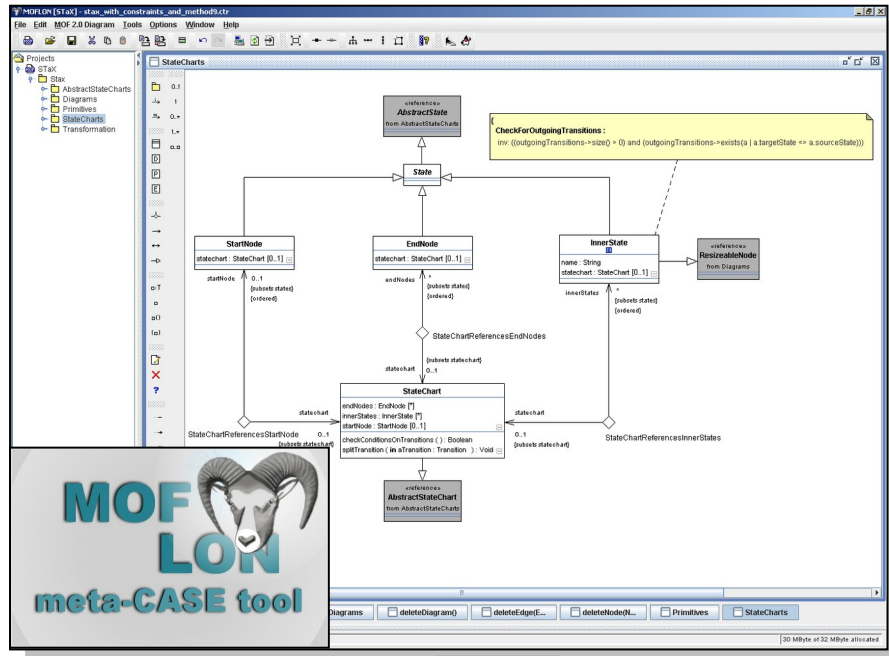
Generated
Code
from
Dresden OCL



Result of MOFLON Example 1 – Statechart Editor (STaX)

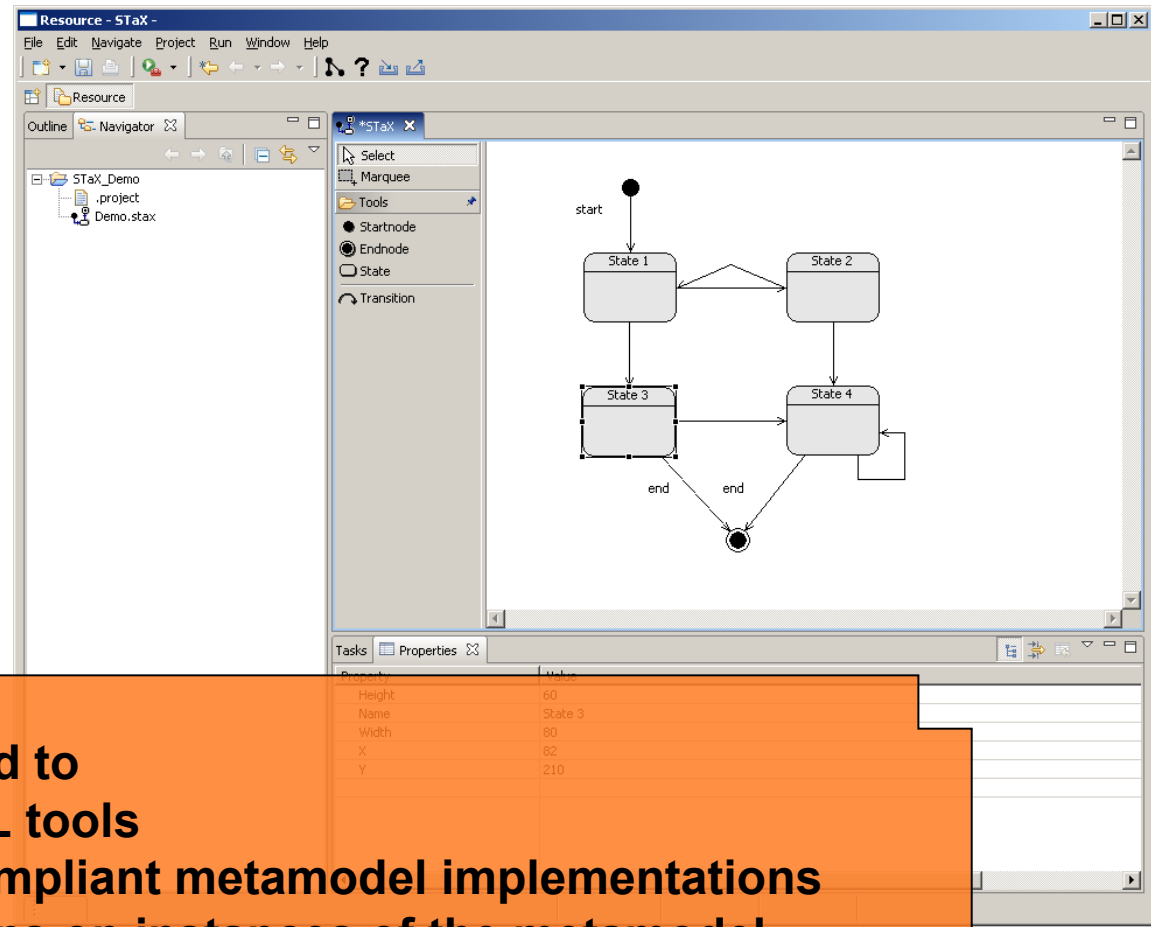
17

Model-Driven Software Development in Technical Spaces (MOST)



Editor:

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



+



© Prof. Dr. ...

MOFLON is mainly used to

- integrate existing DSL tools
- generate standard compliant metamodel implementations
- specify transformations on instances of the metamodel

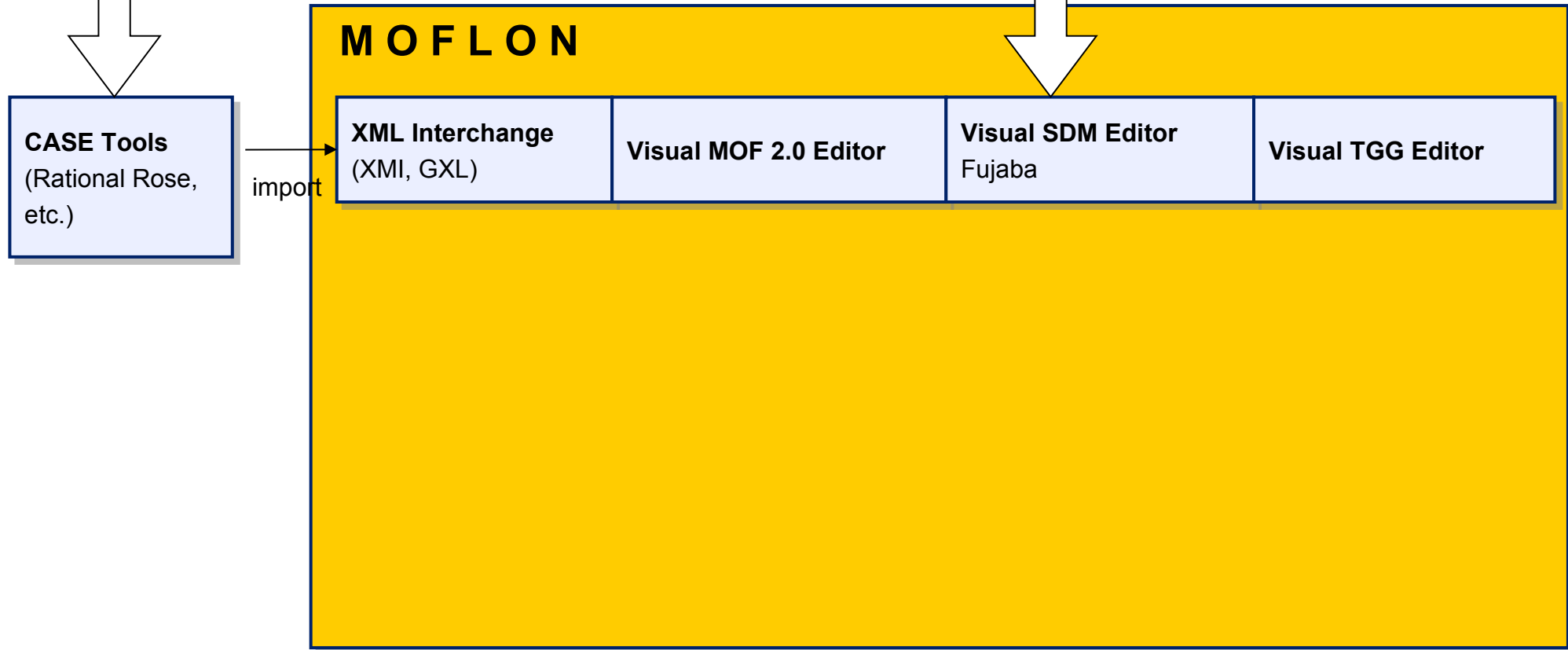


34.3.3 MOFLON – Architecture

18

Model-Driven Software Development in Technical Spaces (MOST)

Obtain Specific Meta Models, Tool Representations



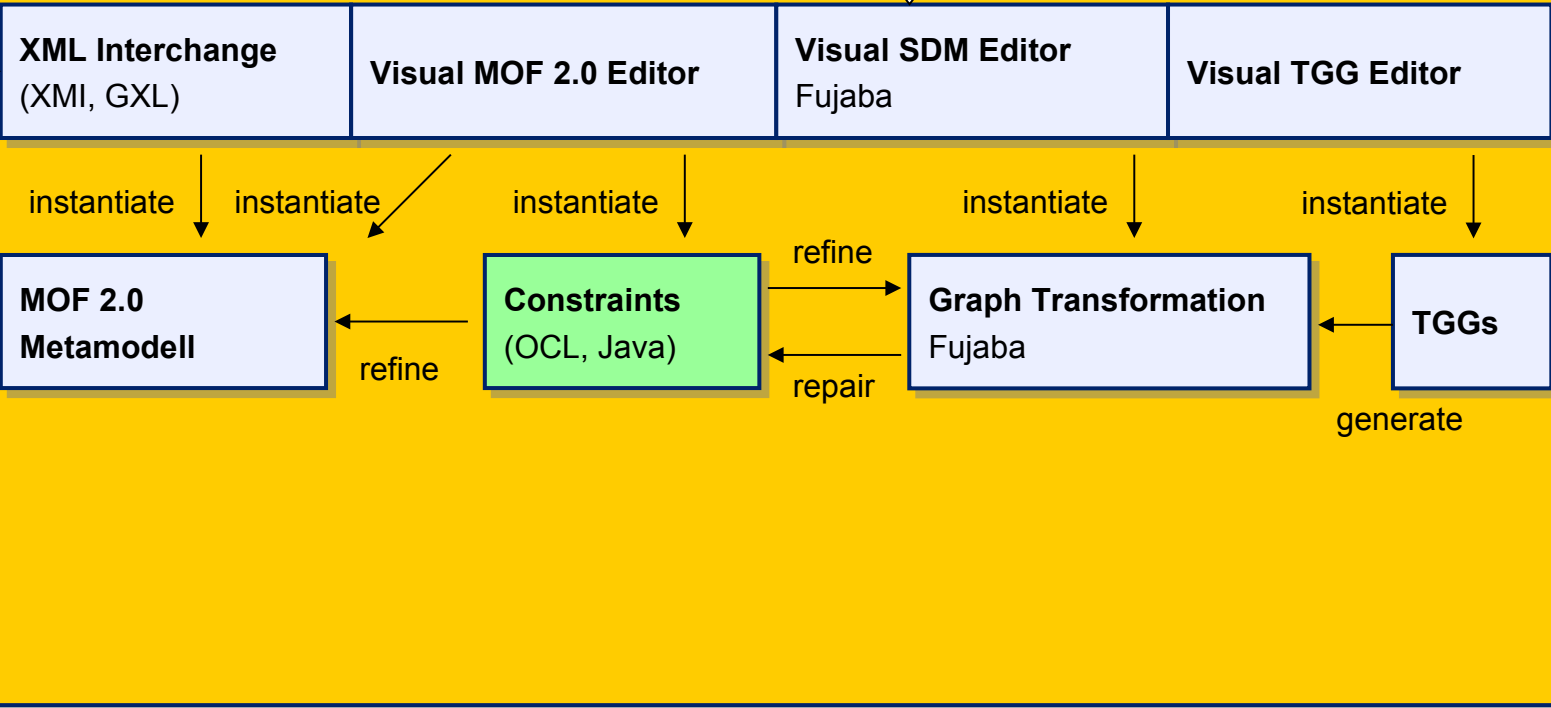
MOFLON - Architecture

Domain Specific Meta Models, Tool Representations

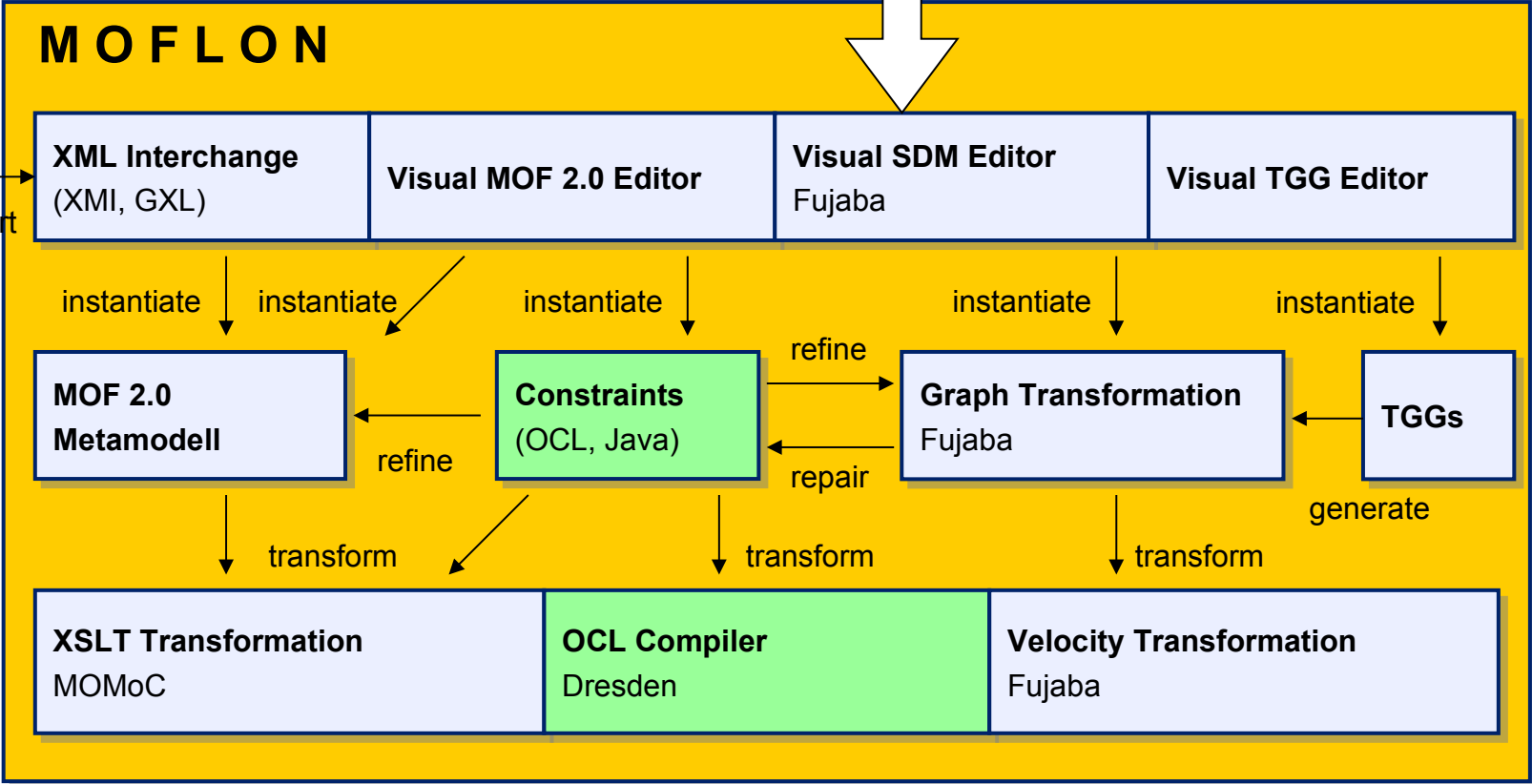
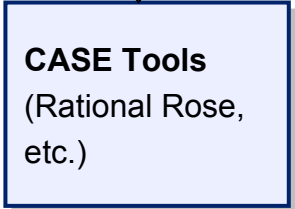
CASE Tools
(Rational Rose,
etc.)

import

MOFLON



MOFLON - Architecture

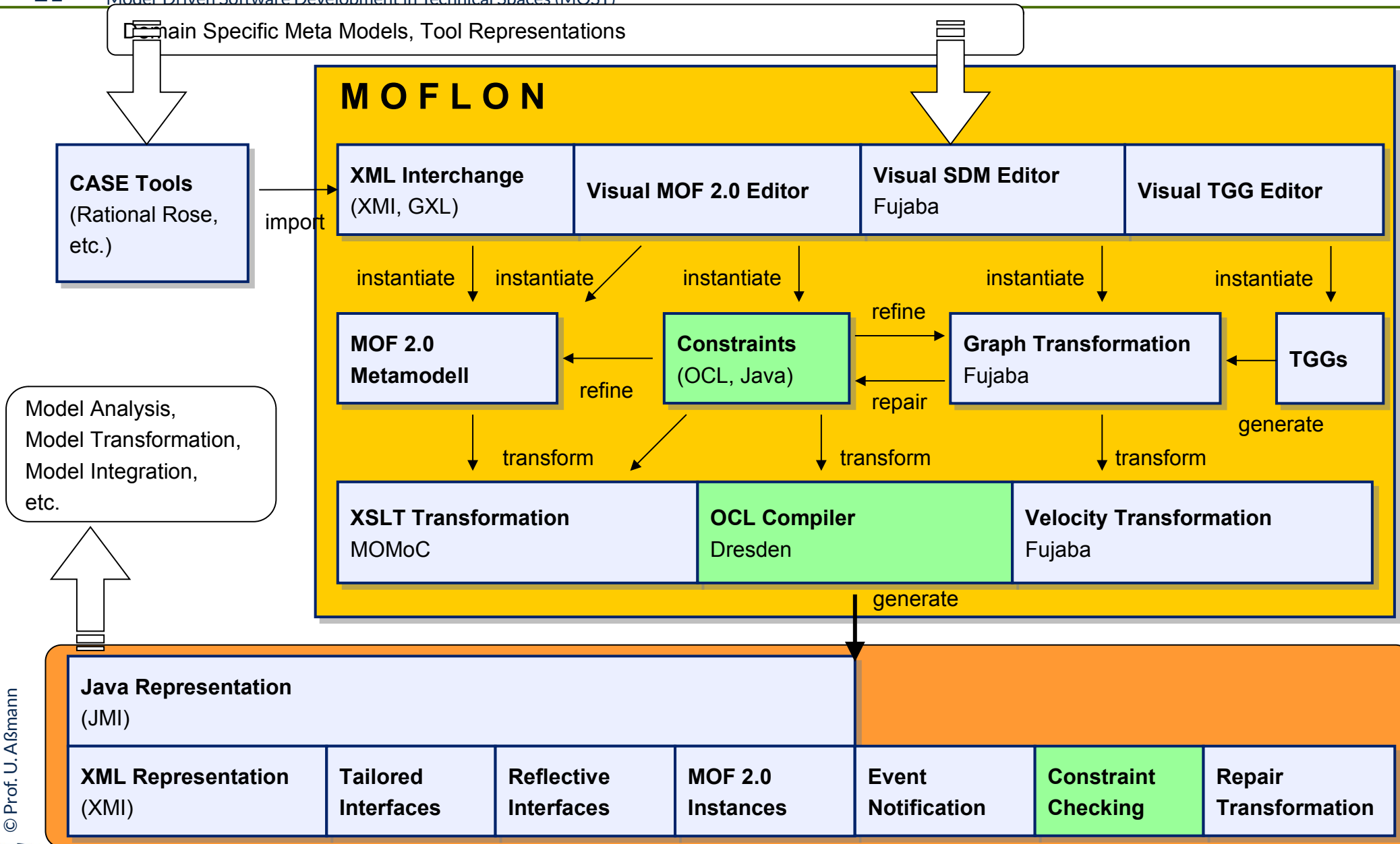


MOFLON – Architecture

21

Model-Driven Software Development in Technical Spaces (MOST)

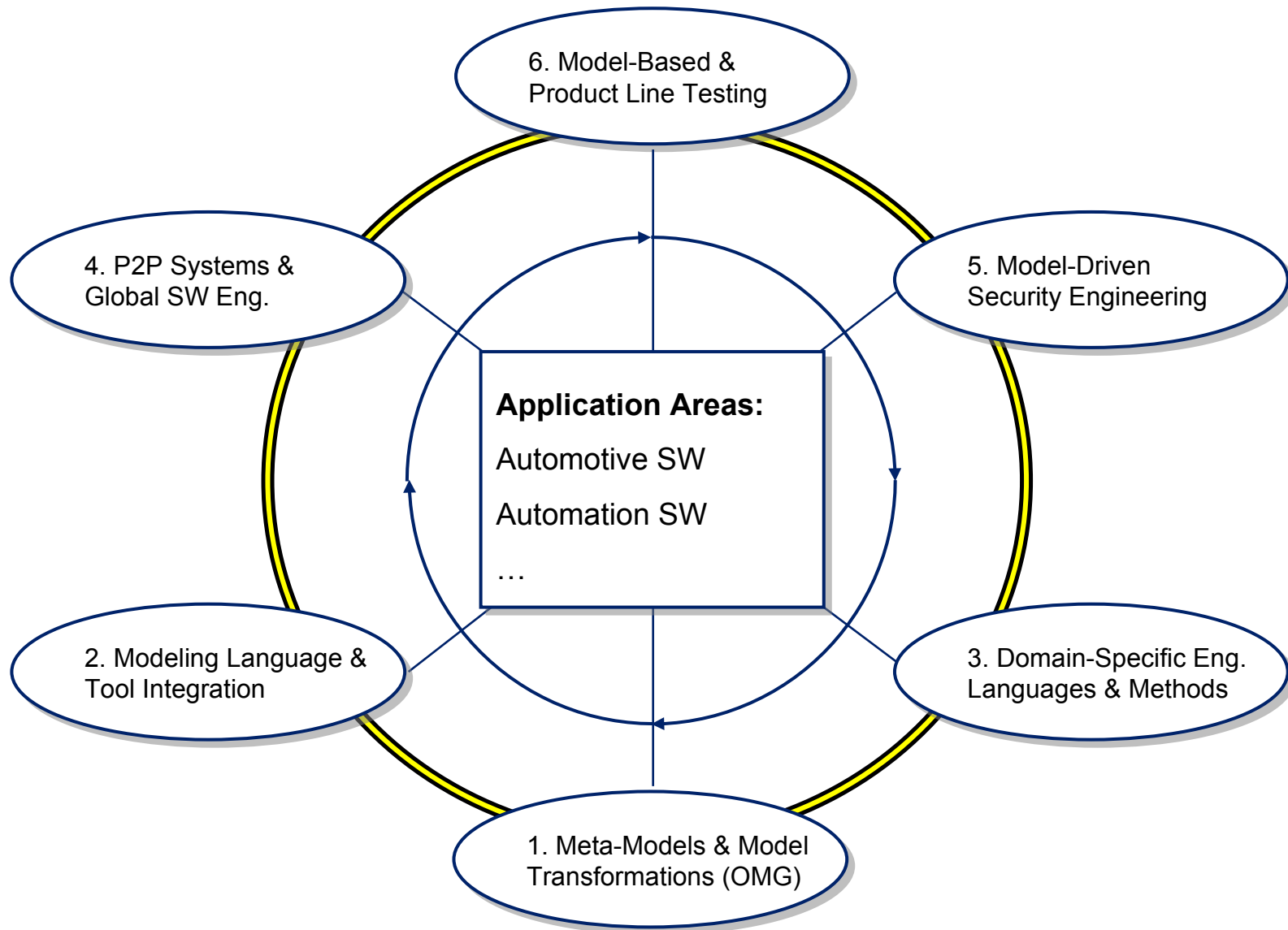
Domain Specific Meta Models, Tool Representations



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	Fujaba & MOFLON	Progres & TGG	GME & TGG	EMF & GReAT	EMF & Tefkat	AToM ³	Microsoft MetaEdit+	EMF & DSL	Pounamu	EBNF & DiaGen	EBNF & TXL	SQL	XML	
Abstract syntax	+	+	+	+	0	0	0	+	+	0	+	+	+	0	+
Concrete syntax	--	--	--	+	+	--	+	+	+	+	+	+	--	--	--
Static semantics	+	+	0	+	+	+	0	0	--	+	0	+	0	0	--
Dynamic semantics	+	+	+	+	+	+	+	0	0	--	--	--	+	--	0
Model analysis	+	+	+	+	+	0	+	0	--	+	--	0	+	0	+
Model transformation	+	+	+	+	+	+	+	0	--	--	--	0	+	0	+
Model integration	+	+	+	+	0	+	--	--	--	--	--	--	0	--	0
Acceptability	+	+	0	--	0	+	--	+	--	0	+	0	0	+	+
Scaleability	+	+	--	0	--	0	--	0	--	--	--	--	--	--	0
Tool availability	--	0	0	+	+	+	+	+	0	0	+	+	+	+	0
Expressiveness	+	+	0	+	+	0	0	0	0	0	0	0	+	0	0

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>

