

43. Das Meta-CASE-Tool MOFLON

1

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und
Multimediatechnik
<http://st.inf.tu-dresden.de>
Version 12-1.1, 05.01.13

1) MOFLON Meta-CASE-
Werkzeug



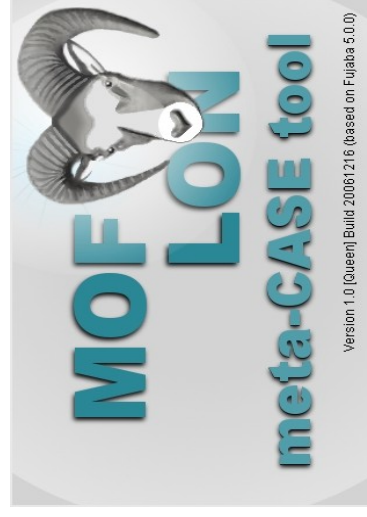
Softwareentwicklungswerkzeuge (SEW) © Prof. Uwe Aßmann

Reading

MOFLON Website
<http://www.moflon.org>

MOFLON Training
<http://moflon.org/documentation/links.html>

MOFLON Tutorial
<http://moflon.org/documentation/tutorial.html>



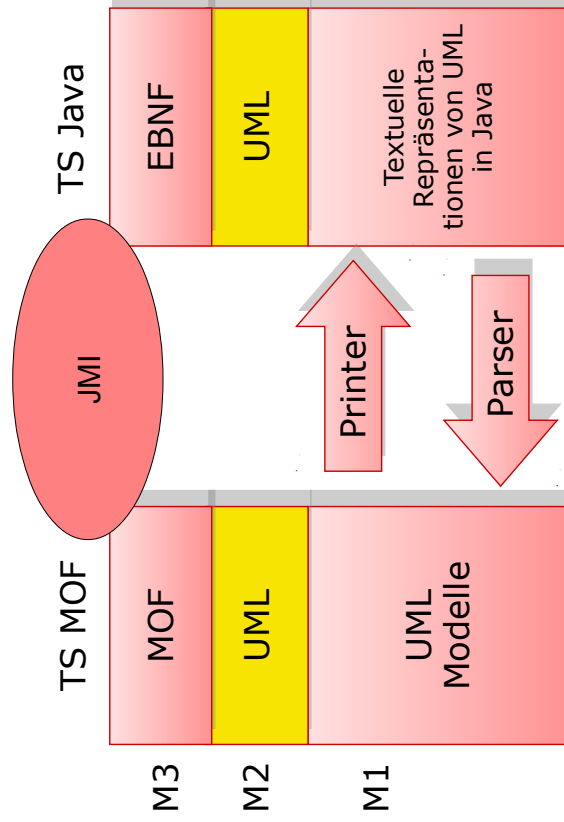
43.3.1. MOFLON Einführung

- ▶ MOFLON ist ein Metamodellierungswerkzeug der TU Darmstadt, Fachgruppe Echtzeitsysteme, Prof. Andy Schürr
 - MOFLON nutzt Logik (OCL) zum Checking von Wohlgeformtheitsbedingungen über Modellen (AC-Werkzeug)
 - MOFLON ist eine Fujaba-Erweiterung und bietet daher Graphersetzungssysteme an www.fujaba.de (M-Werkzeug)
 - MOFLON unterstützt Triple Graph Grammars (TGG, siehe ST-II)
- ▶ MOFLON unterstützt
 - MOF 2.0
 - OCL 2.0
 - JMI 1.4
 - XMI 2.1



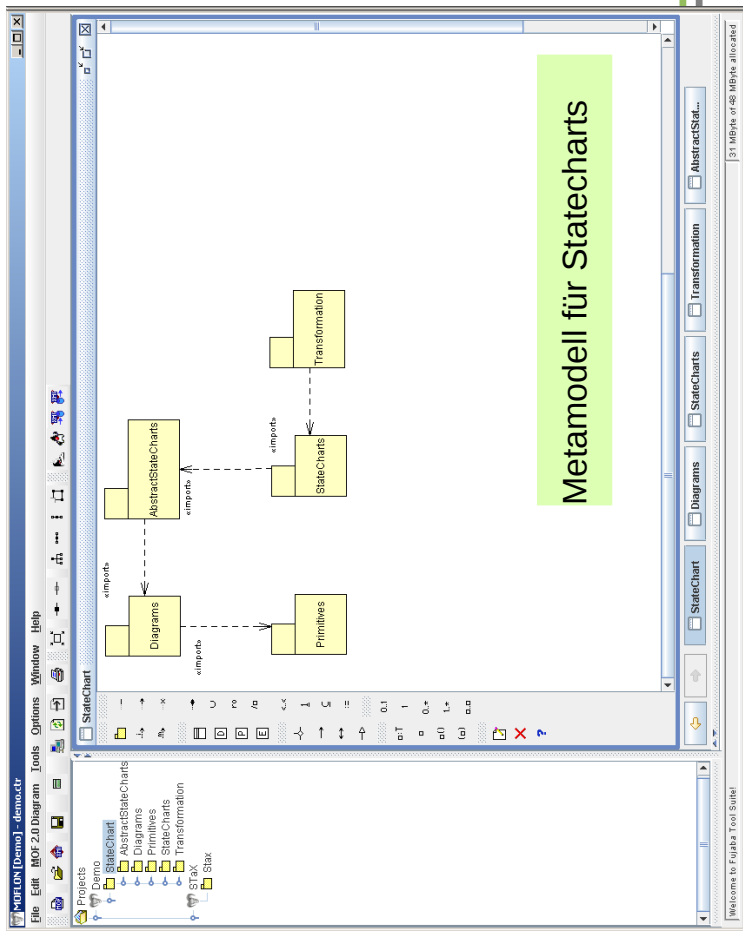
Codegenerierung mit JMI, einer transformative TS-Brücke für MOF und Java, Sprache UML

- ▶ Ähnlich zu XMI, Java Metadata Interchange (JMI) ist eine TS-Halb-Brücke für MOF und EBNF-Space, für die Sprache UML

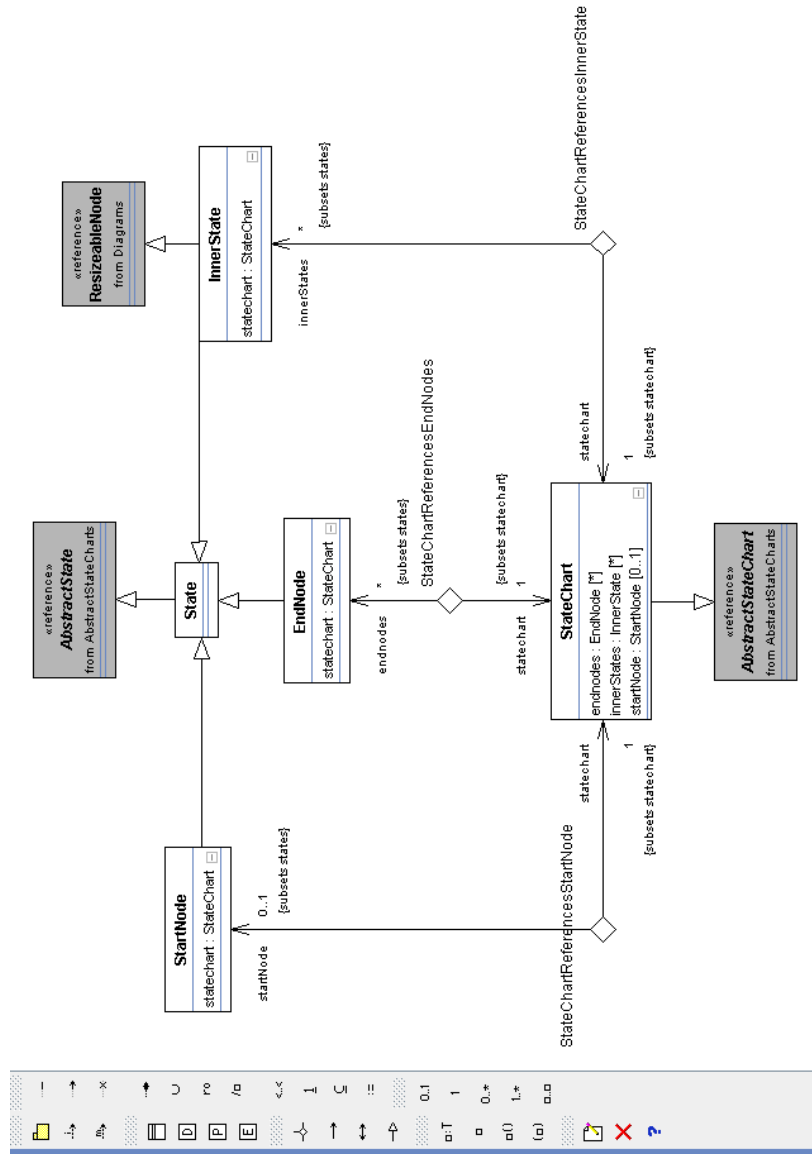


MOFLON Beispiel 1: Metamodell für Statecharts: Vorgehensweise

- 1) Metamodell erstellen
- 2) Code generieren (Repository, Constraint-checker)
- 3) Code über JMI-Schnittstellen verwenden

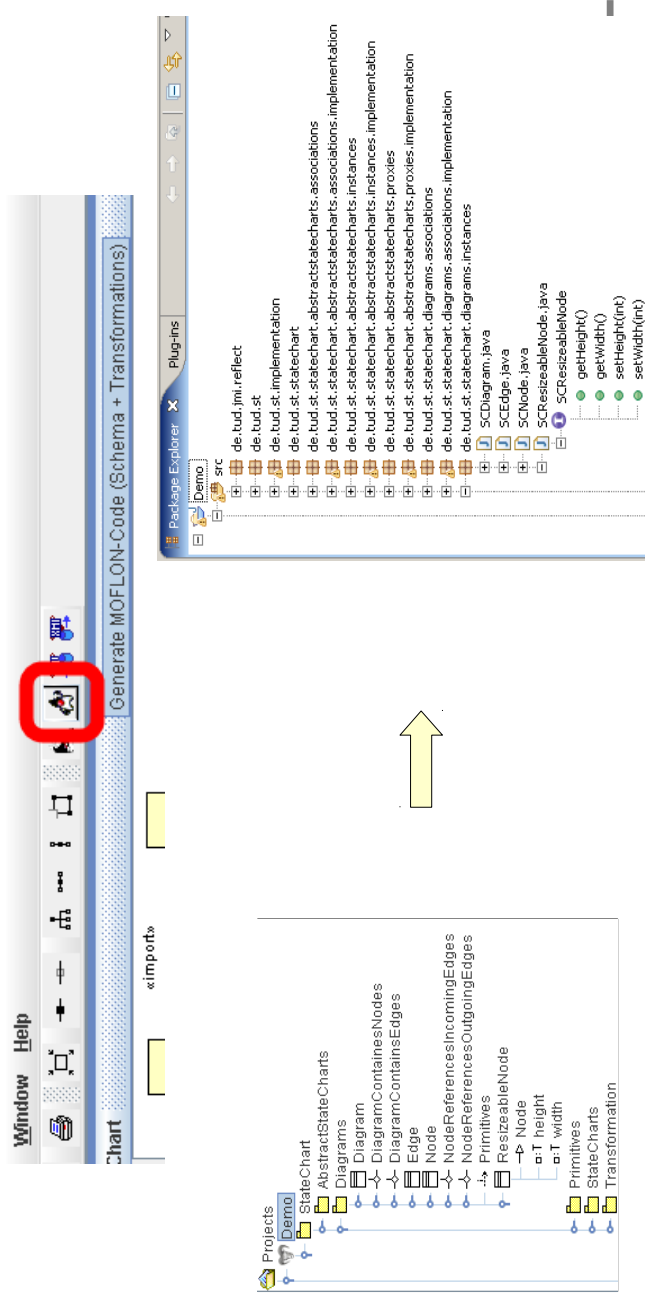


Beispiel: 1.a) Erstellung eines MOF-Metamodells für Statecharts



Beispiel: 1.b) Codegenerierung aus Metamodell für Statechart-Modelle

- ▶ Erzeugt JMI-Schnittstellen zum Metamodell (metamodellgesteuertes Repositoryum)
- ▶ Generiert Code für alle als Story-Diagramm (Fujaba) modellierten Methoden
- ▶ Codegenerator verwendet Velocity und XSLT 1.1



Beispiel: 1.b) Codegenerierung aus Metamodell für Statechart-Modelle

Code generieren

Pro Package

- Java Paket
 - Schnittstelle
 - Implementierung
- de.tud.st.statechart
SCStatechartPackage.java
SCStatechartPackageImpl.java

Pro Klasse

- Schnittstelle
 - Implementierung
 - Proxy Schnittstelle
 - Proxy Implementierung
- SCNode.java
SCNodeImpl.java
SCNodeClass.java
SCNodeClassImpl.java

Pro Assoziation

- Schnittstelle
 - Implementierung
- SCDiagramContainsEdges.java
SCDiagramContainsEdgesImpl.java

Beispiel: 1.c) Codeverwendung von Statechart-Modellen

- ▶ Wurzepaket instanzieren

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

- ▶ Proxy anfordern

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

- ▶ Über den Proxy Instanzen erzeugen

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



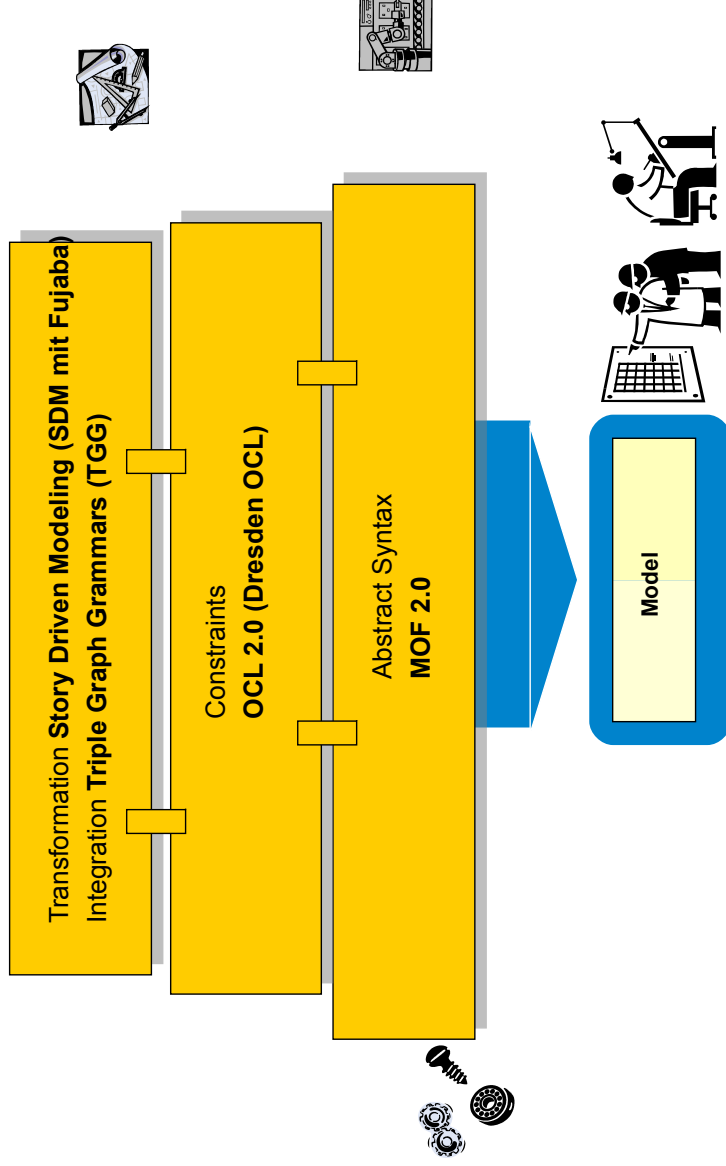
43.3.2. The Metamodeling Architecture of MetaCASE Tool MOFLON



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



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
- ▶ 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

- ▶ 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 shows the 'Edit MOF Constraint' dialog box. The 'General' tab is active, showing the constraint name 'attrNamesMustDiffer', language 'OCL', and body 'inv:attrs->forAll(a1, a2:Attribute|a1<>a2 implies a1.name <> a2.name)'. The 'Visibility' section has 'public' selected. A red dashed arrow points from the 'validate constraints' button to the 'Class' element in the background UML diagram. The 'Class' element in the diagram has a red circle around it, and the 'Attribute' element in the foreground also has a red circle around it.

(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

The screenshot shows three overlapping windows of generated code. The top window, 'MOFLON-code', shows a `reVerifyConstraint(String name): JmiException` method. The middle window, 'Dresden OCL-code', shows OCL constraints for the `Class` class, including `<<calls>>` and `<<queries>>` blocks. The bottom window, 'JMI compliant method', shows the implementation of `reVerifyConstraints(boolean deepVerify): Collection`. A blue arrow points from the 'AttrToType' label in the MOFLON-code to the corresponding OCL code. A red dashed arrow points from the 'validate constraints' button in the MOF Editor to the 'Dresden OCL-code' window.

JMI compliant method

```

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             informalListener(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             informalListener(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

```

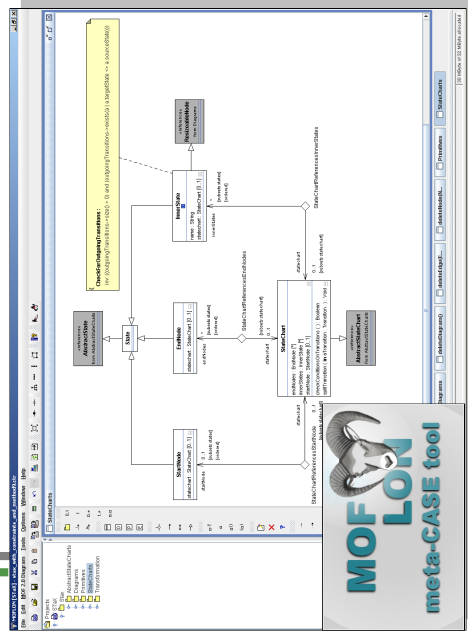
```

348 // generating constraint evaluation method attrNamesMustDiffer
349 public boolean evaluate_attrNamesMustDiffer() {
350     // Variables
351     final tudresden.oc120.core.lib.OclFactory tudOc120Fact0 = tudresden.oc120.core.lib.JmiOclFactory.getInstance(refOutermostPackage());
352     final tudresden.oc120.core.lib.OclCollectionType tudOc120Type1 = tudOc120Fact0.getOclModelTypeFor("cd_metamodel::Attribute");
353     final tudresden.oc120.core.lib.OclPrimitiveType tudOc120Type2 = tudresden.oc120.core.lib.OclPrimitiveType.getOclString();
354     final tudresden.oc120.core.lib.OclModelType tudOc120Type0 = tudOc120Fact0.getOclModelTypeFor("cd_metamodel::Clazz");
355
356     // Invariant
357     final tudresden.oc120.core.lib.OclModelObject tudOc120Var0 = (tudresden.oc120.core.lib.OclModelObject) tudOc120Fact0.getOclRepresentationFor(
358         tudOc120Type0, this);
359     final tudresden.oc120.core.lib.OclBag tudOc120Exp0 = tudresden.oc120.core.lib.Ocl.tocOc120Exp0.getFeatures(tudOc120Type1, "attrs");
360     final tudresden.oc120.core.lib.OclIterator tudOc120Iter0 = tudOc120Exp0.getIterator();
361     final tudresden.oc120.core.lib.OclBooleanEvaluatable tudOc120EVal0 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
362     public tudresden.oc120.core.lib.OclBoolean evaluate() {
363         final tudresden.oc120.core.lib.OclModelObject tudOc120Var1 = tudresden.oc120.core.lib.Ocl.tocOc120Iter0.getObject(tudOc120Type0.getValue());
364         final tudresden.oc120.core.lib.OclIterator tudOc120Iter1 = tudOc120Exp0.getIterator();
365         final tudresden.oc120.core.lib.OclBooleanEvaluatable tudOc120EVal1 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
366         public tudresden.oc120.core.lib.OclBoolean evaluate() {
367             final tudresden.oc120.core.lib.OclBoolean evaluate() {
368                 .tocOc120ModelObject(tudOc120Iter1.getValue());
369
370             //T000: Check if VariableId is correct
371             final tudresden.oc120.core.lib.OclBoolean tudOc120Exp1 = tudOc120Var2.isNotEqualTo(tudOc120Var1);
372             final tudresden.oc120.core.lib.OclString tudOc120Exp2 = tudresden.oc120.core.lib.Ocl.tocOc120Exp2;
373             tudOc120Var2.getFeature(tudOc120Type2, "name");
374             final tudresden.oc120.core.lib.OclString tudOc120Exp3 = tudresden.oc120.core.lib.Ocl.tocOc120Exp3;
375             tudOc120Var1.getFeature(tudOc120Type2, "name");
376             final tudresden.oc120.core.lib.OclBoolean tudOc120Exp4 = tudOc120Exp2.isNotEqualTo(tudOc120Exp3);
377             final tudresden.oc120.core.lib.OclBoolean tudOc120Exp5 = tudOc120Exp1.implies(tudOc120Exp4);
378
379             return tudOc120Exp5;
380         }
381     };
382
383     final tudresden.oc120.core.lib.OclBoolean tudOc120Exp6 = (tudresden.oc120.core.lib.OclBoolean) tudOc120Exp0.forAll(
384         tudOc120Iter1, tudOc120EVal1);
385
386     return tudOc120Exp6;
387 };
388
389 final tudresden.oc120.core.lib.OclBoolean tudOc120Exp7 = (tudresden.oc120.core.lib.OclBoolean) tudOc120Iter0.forAll(tudOc120EVal0);
390
391 return tudOc120Exp7.isTrue();
392
393 }

```

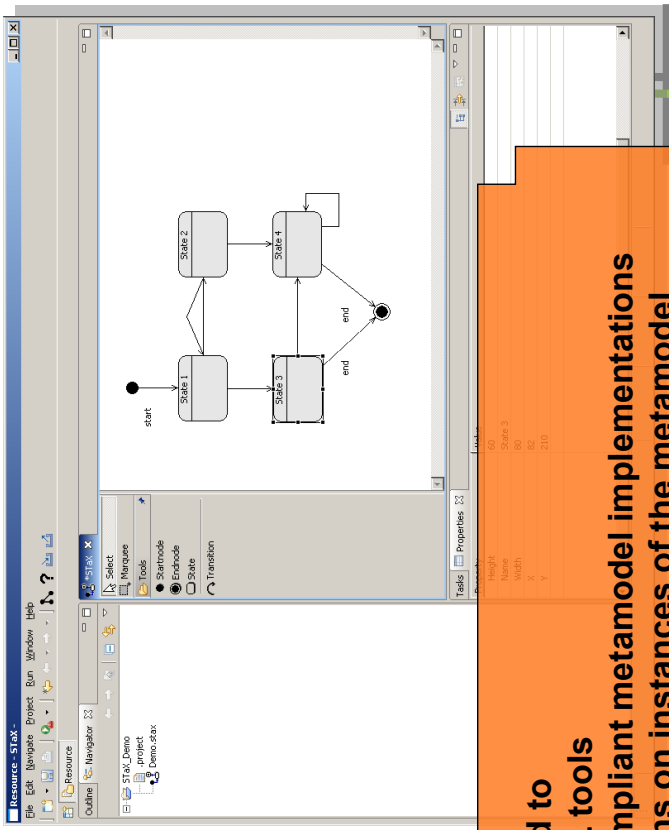
Generated Code from Dresden OCL

Result of MOFLON Example 1 – Statechart Editor (STaX)



Editor:

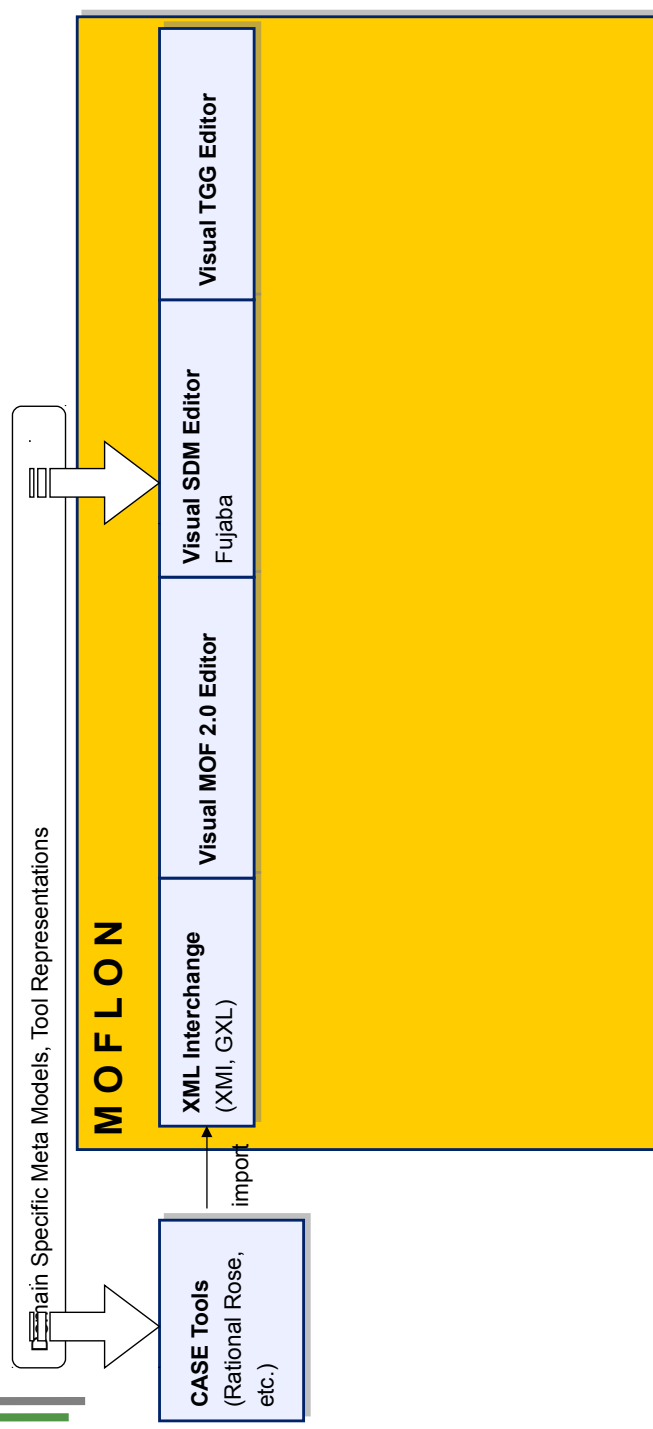
- data structure (MOFLON repository)
- GUI (GEF)



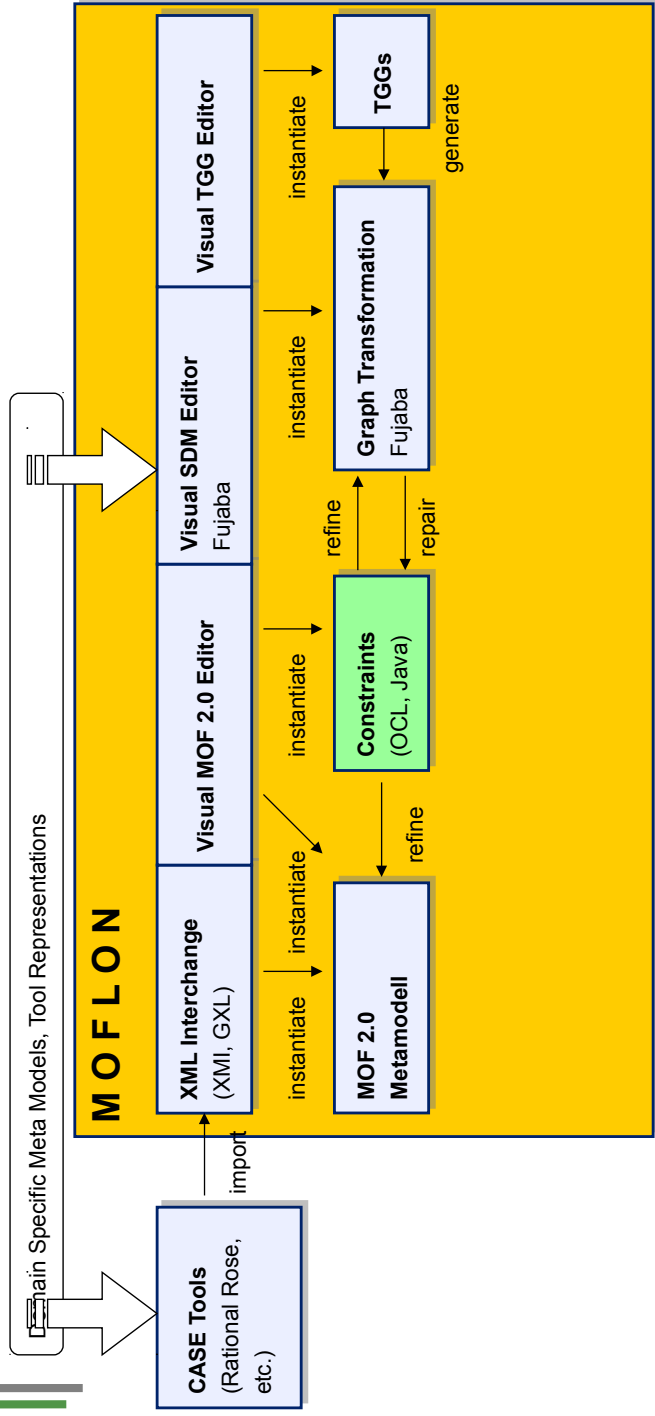
- MOFLON is mainly used to**
- integrate existing DSL tools
 - generate standard compliant metamodel implementations
 - specify transformations on instances of the metamodel



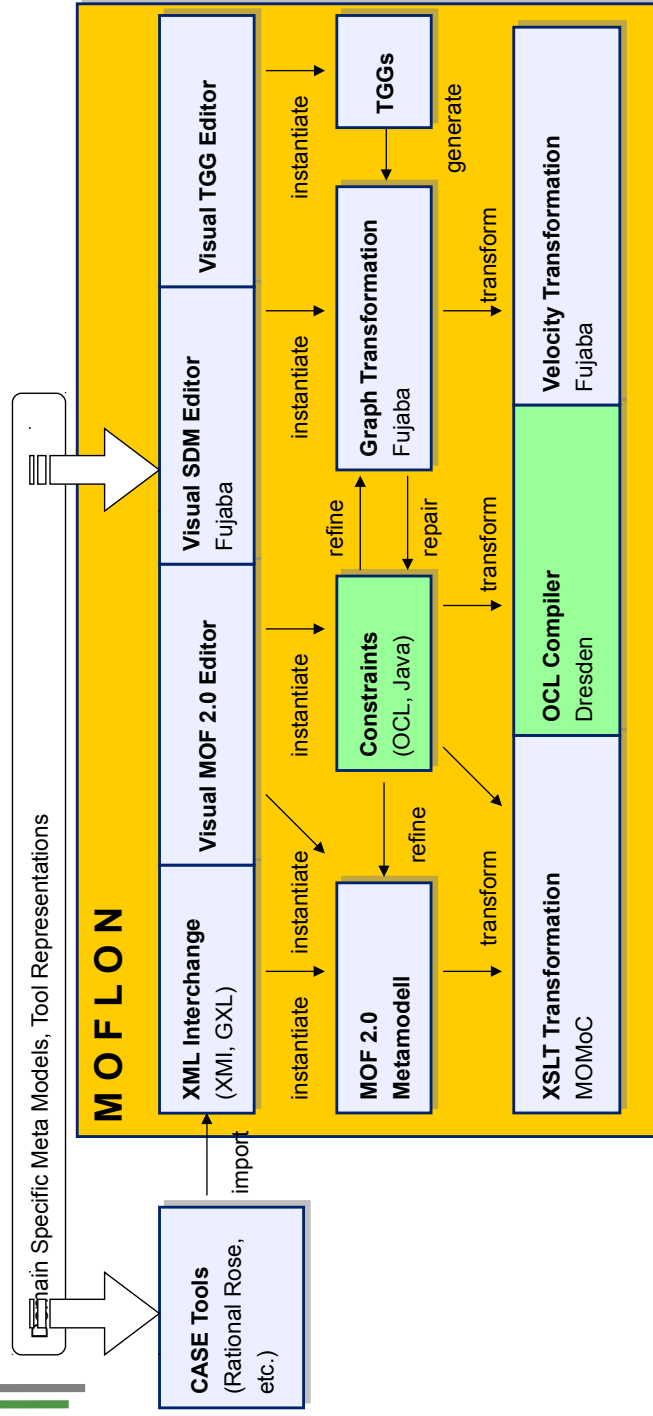
43.3.3 MOFLON – Architecture



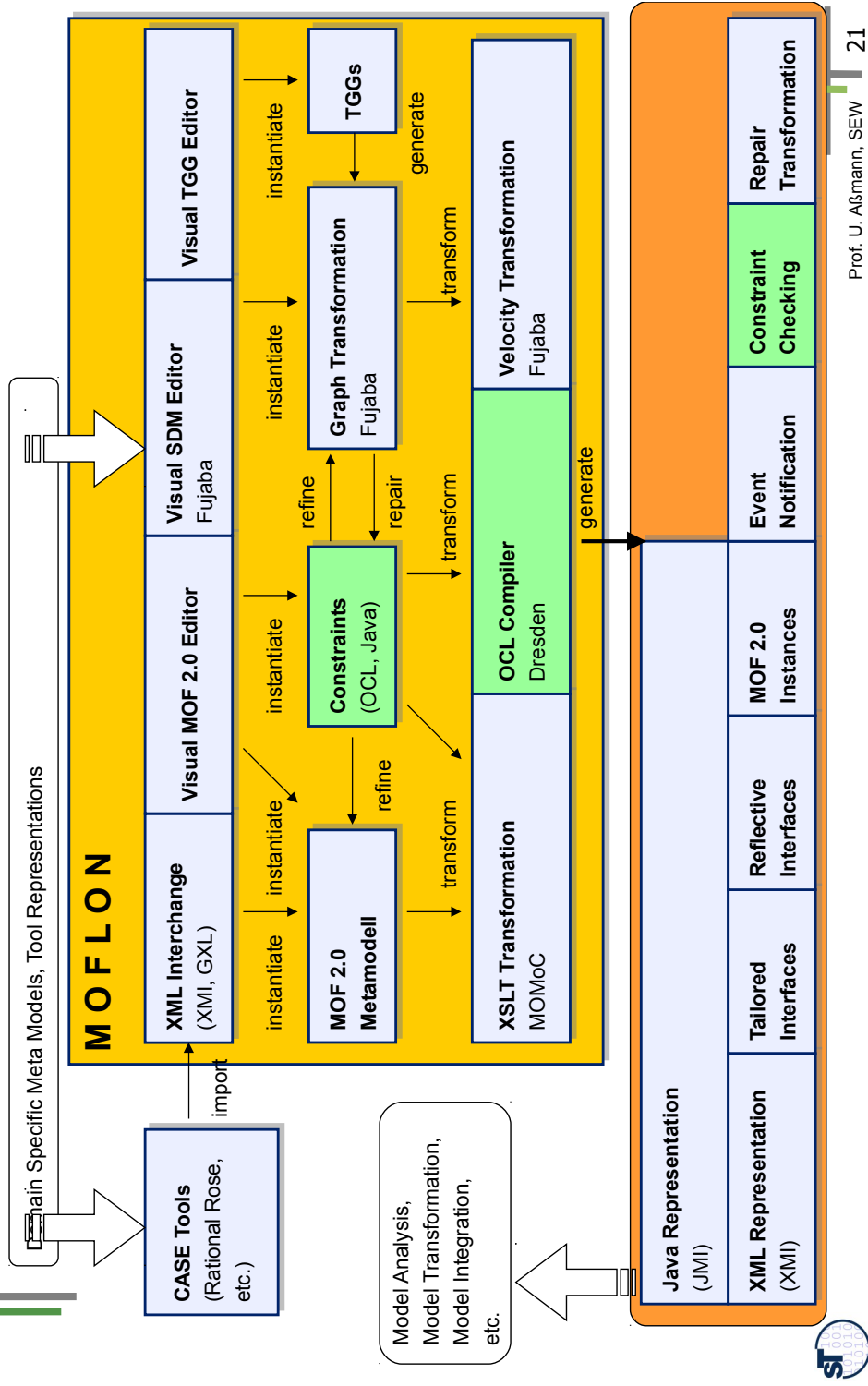
MOFLON – Architecture



MOFLON – Architecture

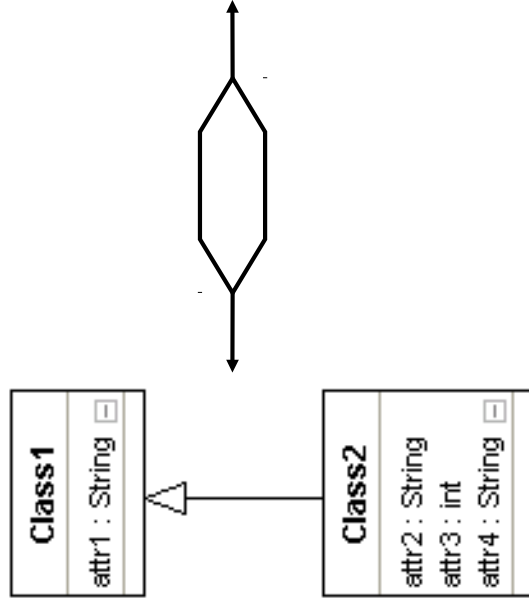


MOFLON – Architecture



43.3.4 Example 2: Integration with TGG - Object-Relational Mapping (ORM) from Class Diagrams to Database Schema

domain specific language, e.g. Class Diagrams

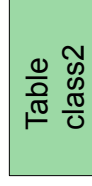


domain specific language, e.g. Database Schemata

Table class1

Server: localhost Database: icgt2008 Table: class1

Field	Type	Collation	Attributes	Null
attr1	varchar(1024)	latin1_general_ci		No
attr2	varchar(1024)	latin1_general_ci		No
attr3	int(11)			No
attr4	varchar(1024)	latin1_general_ci		No

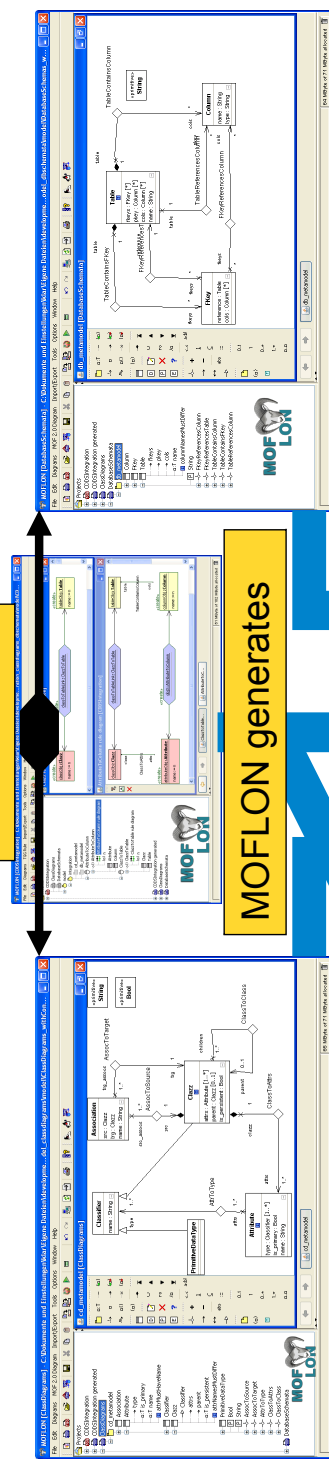


Example 2: Tool Integration Scenario TIE-CDDS: (ClassDiagrams / DatabaseSchema)

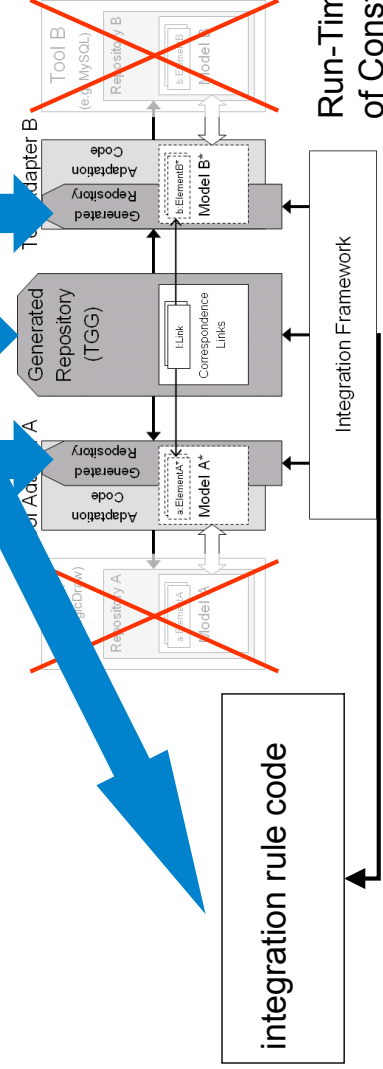
Class Diagrams Metamodel

Database Schemata Metamodel

TGGs relate



MOFLON generates



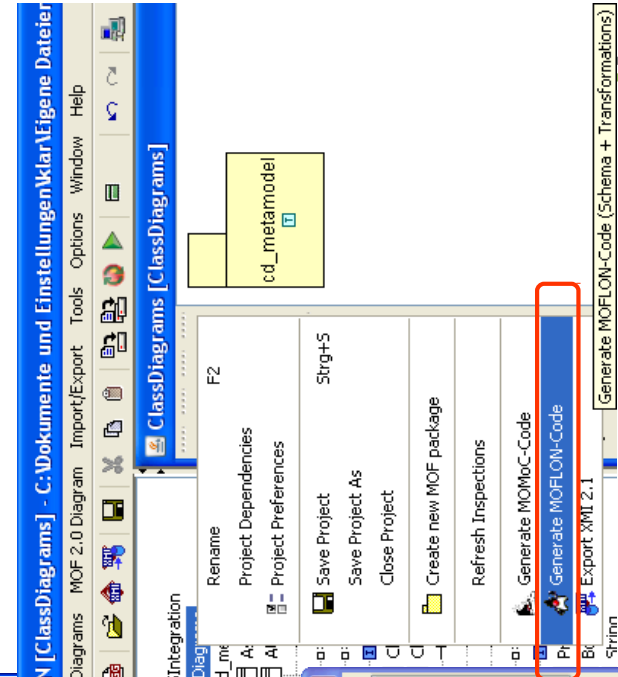
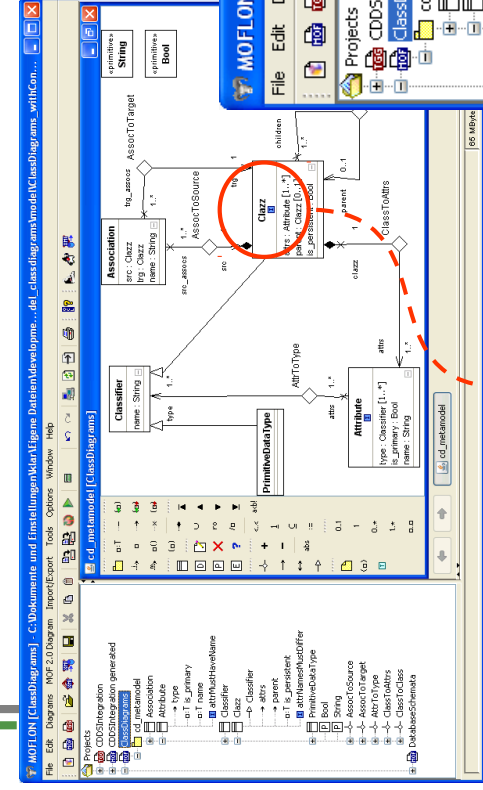
integration rule code

Run-Time Verification of Constraints



Prof. U. Asmann, SEW 23

TIE-CDDS – Constraints in Class Diagrams (1) Generate Code from MOF model (CD metamodel)



Prof. U. Asmann, SEW 24

TiE-CDDS – Constraints in Class Diagrams (2)

Integration Framework

Constraint Validation

source domain model does not fulfill its constraints:
 constraint named 'attrNamesMustDiffer' is violated in instance: Customer: inv:attrs->forall(a1,a2:Attribute|a1.<>a2.implies(a1.name <> a2.name)
 constraint named 'attrMustHaveName' is violated in instance: : inv:name.size()>0
 association 'cd_metamodel.ClassToAttrs', memberEnd 'attrs': size of links is out of bounds in context: 'Order:cd_metamodel.Class': should be [1,unbounded[]] but is 0; inv: attrs->size()>=1 and attrs->size()<=unbounded

model violates constraints:

- class „Customer“ has two attributes with same name: „name“
- attribute in class „Address“ has no name
- multiplicity violation: class „Order“ has no attribute but according to CD metamodel every class must have one

visualization of classdiagrams model (here: source domain)

```

classDiagram
    class Address {
        name : AttributeImpl
    }
    class Customer {
        name : AttributeImpl
    }
    class Order {
        name : AttributeImpl
    }
    class String {
        name : PrimitiveDataImpl
    }
    class Association {
        name : AssociationImpl
    }
    class PrimitiveData {
        name : PrimitiveDataImpl
    }
    Address "1" -- "*" Customer
    Customer "1" -- "*" Order
    Order "1" -- "*" String
    Association "1" -- "*" PrimitiveData
  
```

TiE-CDDS – Constraints in Class Diagrams (3)

Model Browser

Model Browser

Model structure:

- cd_metamodel
 - customer: AssociationImpl
 - address: AssociationImpl
 - Order: ClassImpl
 - id: AttributeImpl
 - Customer: ClassImpl
 - surname: AttributeImpl
 - name: ClassImpl
 - Address: AttributeImpl
 - street: AttributeImpl
 - int: PrimitiveDataImpl
 - String: PrimitiveDataImpl

String Editor Dialog

Change value...
 surname

model is fixed in generic model editor

TiE-CDDS – Constraints in Class Diagrams (4)

Integration Framework

The screenshot shows the TiE-Integration Framework interface. The 'Tool Adapter' section is expanded, showing options for 'ini_adapter_classdiagrams_offline.jar' and 'ini_adapter_observable_offline.jar'. The 'Link Domain' section is also expanded, showing 'integration_classdiagrams_observable.jar' and 'cdds_observable.xml'. The 'Configuration File' is set to 'C:\offline\Offline.conf'. The 'Algorithm' is set to 'Forward Translation (Batch, Simple)'. The 'Log Level' is set to 'WARN'. The 'Output' section shows 'root' and 'SOURCE'.

translation process
may start now...

Constraint Validation

source domain model fulfills its constraints

OK

TiE-CDDS – Constraints in Class Diagrams (5)

Forward Translation to DB representation

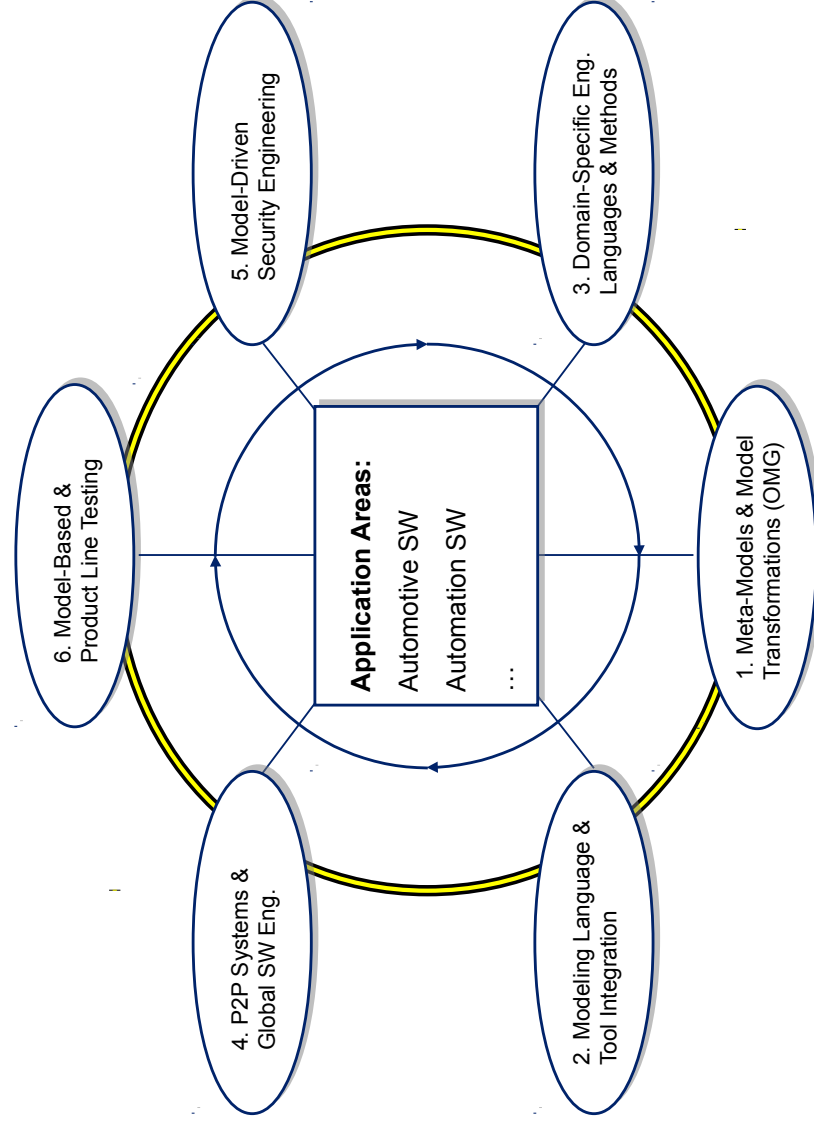
The screenshot shows the TiE-Integration Framework interface. The 'Output' section is expanded, showing 'root' and 'SOURCE'. The 'LinkBrowser' section is expanded, showing a tree view of the source domain model. The 'Log' section shows the output of the forward translation, including the generation of the database schema.

The screenshot shows the TiE-Integration Framework interface. The 'Output' section is expanded, showing 'root' and 'SOURCE'. The 'LinkBrowser' section is expanded, showing a tree view of the source domain model. The 'Log' section shows the output of the forward translation, including the generation of the database schema. The 'Action' section is expanded, showing 'Forward Translation (Batch, Simple)'. The 'Log Level' is set to 'WARN'. The 'Output' section shows 'root' and 'SOURCE'. The 'LinkBrowser' section is expanded, showing a tree view of the source domain model. The 'Log' section shows the output of the forward translation, including the generation of the database schema.

Future Work – OCL

- ▶ We bootstrap our MOFLON MOF Metamodel periodically
 - Add more OCL constraints to our MOF Metamodel
 - Regenerate MOFLON MOF implementation
 - Activate 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	AToM ³	MetaEdit+	Microsoft DSL	EMF & GMF	Pounamu	DiaGen	EBNF & TXL	SQL	XML
Abstract syntax	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Concrete syntax	--	--	--	+	+	+	--	+	+	+	+	+	--	--	--
Static semantics	+	+	+	+	+	+	+	+	+	--	+	+	+	+	--
Dynamic semantics	+	+	+	+	+	+	+	+	+	+	--	--	+	+	+
Model analysis	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Model transformation	+	+	+	+	+	+	+	+	+	--	--	--	+	+	+
Model integration	+	+	+	+	+	+	+	--	--	--	--	--	+	+	+
Acceptability	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Scaleability	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Tool availability	--	--	--	+	+	+	+	+	+	+	+	+	+	+	+
Expressiveness	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

from Amelunxen, Königs, Röttschke, 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 Modeling 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. Röttschke, 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.moflon.org>

