

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 13-0.1, 02.01.14

1) MOFLON Meta-CASE-
 Werkzeug

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_AGTIME_2011.pdf
- ▶ MOFLON Training
 - <http://moflon.org/documentation/links.html>
- ▶ MOFLON Tutorial
 - <http://moflon.org/documentation/tutorial.html>

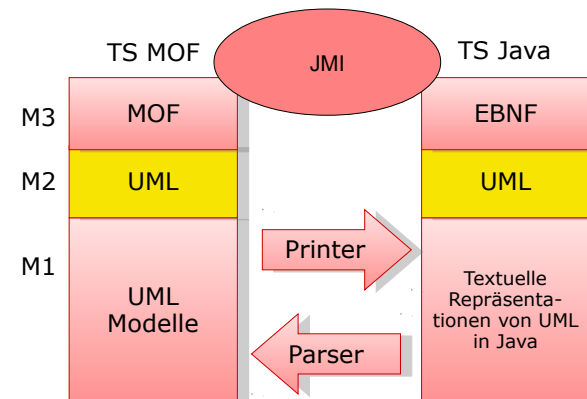


43.3.1. MOFLON Einführung

- ▶ 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

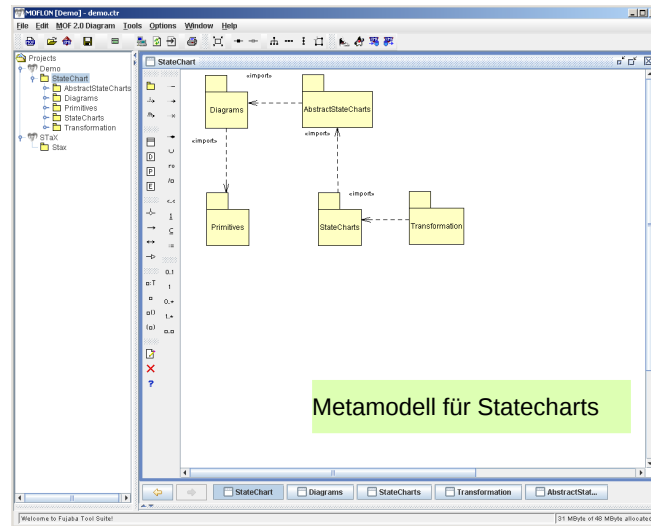
Codegenerierung mit JMI, einer transformative TS-Brücke für MOF und Java, Sprache UML

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

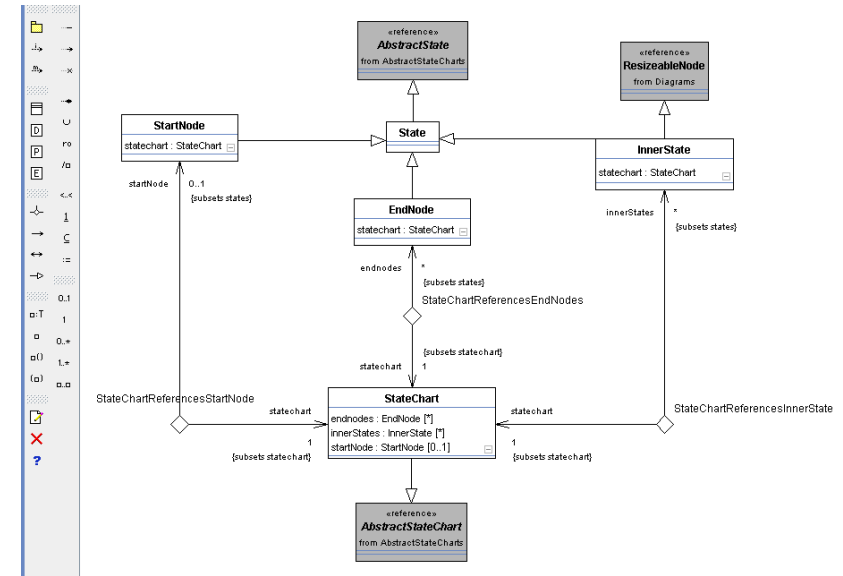


MOFLON Beispiel 1: Metamodell für Statecharts: Vorgehensweise

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

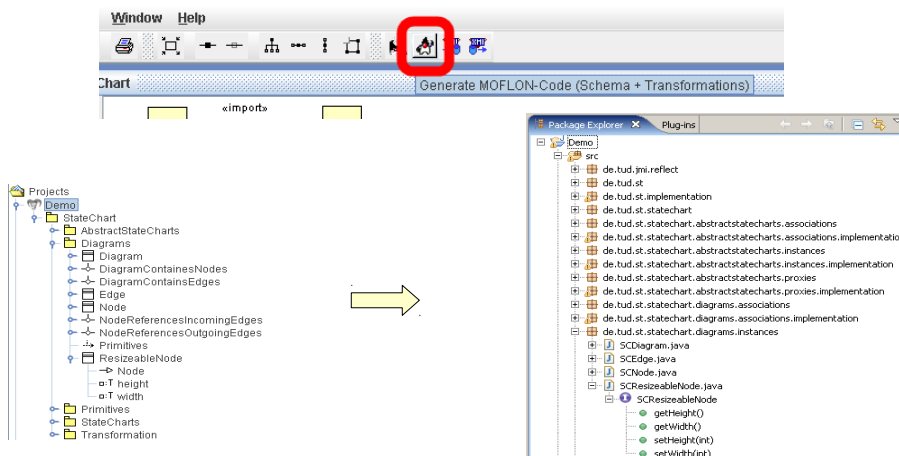


Beispiel: 1.a) Erstellung eines MOF-Metamodells für Statecharts



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

- Erzeugt JMI-Schnittstellen zum Metamodell (metamodellgesteuertes Repository)
- 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: de.tud.st.statechart
- Schnittstelle: SCStateChartPackage.java
- Implementierung: SCStateChartPackageImpl.java

Pro Klasse

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

Pro Assoziation

- Schnittstelle: SCDiagramContainsEdges.java
- Implementierung: SCDiagramContainsEdgesImpl.java

Beispiel: 1.c) Codeverwendung von Statechart-Modellen

- ▶ Wurzepaket instanzieren

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

- ▶ Proxy anfordern

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

- ▶ Über den Proxy Instanzen des Modells 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



ES Real-Time Systems Lab

Prof. Dr. rer. nat. Andy Schürr

Dept. of Electrical Engineering and Information Technology

Dept. of Computer Science (adjunct Professor)

www.es.tu-darmstadt.de

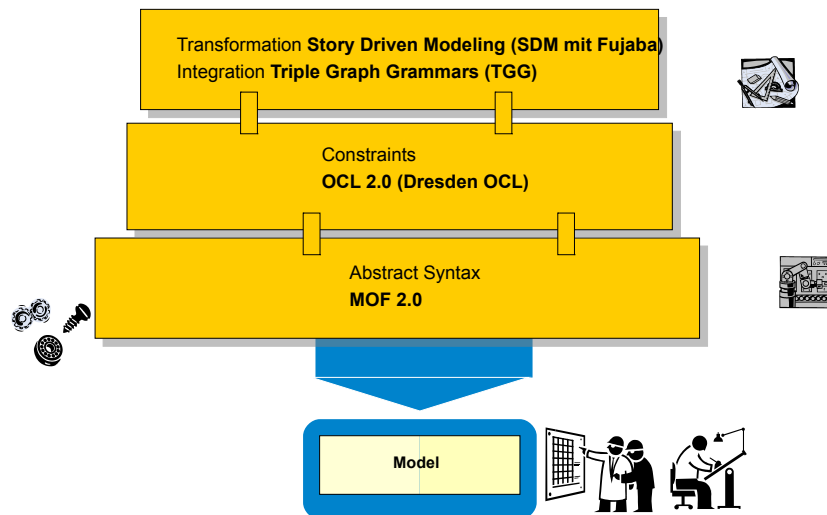
Felix Klar

Felix.Klar@es.tu-darmstadt.de

© author(s) of these slides 2009 including research results of the research network ES and TU Darmstadt otherwise as specified at the respective slide

15.10.2009

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

Edit MOF Constraint

Name: attrNamesMustDiffer
Language: OCL
Body: inv:attrs->forall(a1,a2:Attribute|a1<>a2 implies a1.name <> a2.name)

validate constraints

(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

MOFLON-code
refVerifyConstraint(String name):JmiException

Dresden OCL-code

generated Repository
Model A
c1.Clazz

JMI compliant method
refVerifyConstraints(boolean deepVerify):Collection

JMI compliant method

```

public Collection<String> refVerifyConstraints(boolean deepVerify) {
    Collection<String> constraintNames = new org.apache.commons.collections.impl.SetImpl<>();

    for (String constraintName : refConstraintNames()) {
        javax.jmi.reflect.JmiException constraintException = refVerifyConstraint(constraintName);

        if (constraintException != null) {
            invalidConstraints.add(constraintException);
        }
    }

    if (deepVerify) {
        if (invalidConstraints.size() > 0) {
            return invalidConstraints;
        } else {
            return null;
        }
    }
}
    
```

Generated Code from Dresden OCL

```

// generating constraint evaluation method attrNamesMustDiffer
public boolean evaluate_attrNamesMustDiffer() {
    // Variables
    final tudresden.oc120.core.lib.JmiOclFactory tud0c120Fact0 = tudresden.oc120.core.lib.JmiOclFactory.getInstance(refOutermostPackage());
    final tudresden.oc120.core.lib.OclCollectionType tud0c120Type1 = tudresden.oc120.core.lib.OclCollectionTypeFor("ocl_getamodel::Attribute");
    final tudresden.oc120.core.lib.OclPrimitiveType tud0c120Type2 = tudresden.oc120.core.lib.OclPrimitiveTypeFor("ocl1String");
    final tudresden.oc120.core.lib.OclModelType tud0c120Type0 = tud0c120Fact0.getOclModelTypeFor("ocl_getamodel::Clazz");

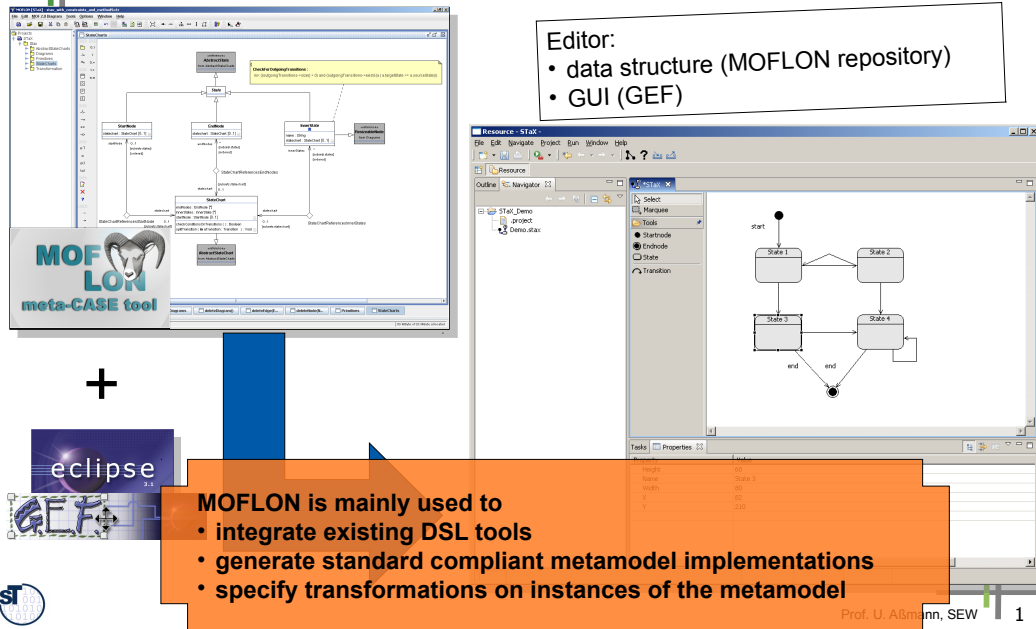
    // Invariant
    final tudresden.oc120.core.lib.OclModelObject tud0c120Var0 = (tudresden.oc120.core.lib.OclModelObject) tud0c120Fact0.getOclRepresentationFor(
        tud0c120Type0, this);
    final tudresden.oc120.core.lib.OclBag tud0c120Exp0 = tudresden.oc120.core.lib.Ocl.toOclBag(tud0c120Var0.getFeature(tud0c120Type1, "attrs"));
    final tudresden.oc120.core.lib.OclIterator tud0c120Iter0 = tud0c120Exp0.getIterator();
    final tudresden.oc120.core.lib.OclBooleanEvaluatable tud0c120Eval0 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
        public tudresden.oc120.core.lib.OclBoolean evaluate() {
            final tudresden.oc120.core.lib.OclModelObject tud0c120Var1 = tudresden.oc120.core.lib.Ocl.toOclModelObject(tud0c120Iter0.getValue());
            final tudresden.oc120.core.lib.OclIterator tud0c120Iter1 = tud0c120Exp0.getIterator();
            final tudresden.oc120.core.lib.OclBooleanEvaluatable tud0c120Eval1 = new tudresden.oc120.core.lib.OclBooleanEvaluatable() {
                public tudresden.oc120.core.lib.OclBoolean evaluate() {
                    final tudresden.oc120.core.lib.OclModelObject tud0c120Var2 = tudresden.oc120.core.lib.Ocl
                        .toOclModelObject(tud0c120Iter1.getValue());
                    //TODO: Check if VariableId is correct
                    final tudresden.oc120.core.lib.OclBoolean tud0c120Exp1 = tud0c120Var2.isNotEqualTo(tud0c120Var1);
                    final tudresden.oc120.core.lib.OclString tud0c120Exp2 = tudresden.oc120.core.lib.Ocl.toOclString(
                        tud0c120Var2.getFeature(tud0c120Type2, "name"));
                    final tudresden.oc120.core.lib.OclString tud0c120Exp3 = tudresden.oc120.core.lib.Ocl.toOclString(
                        tud0c120Var1.getFeature(tud0c120Type2, "name"));
                    final tudresden.oc120.core.lib.OclBoolean tud0c120Exp4 = tud0c120Exp2.isNotEqualTo(tud0c120Exp3);
                    final tudresden.oc120.core.lib.OclBoolean tud0c120Exp5 = tud0c120Exp1.implies(tud0c120Exp4);

                    return tud0c120Exp5;
                }
            };
        }
    };

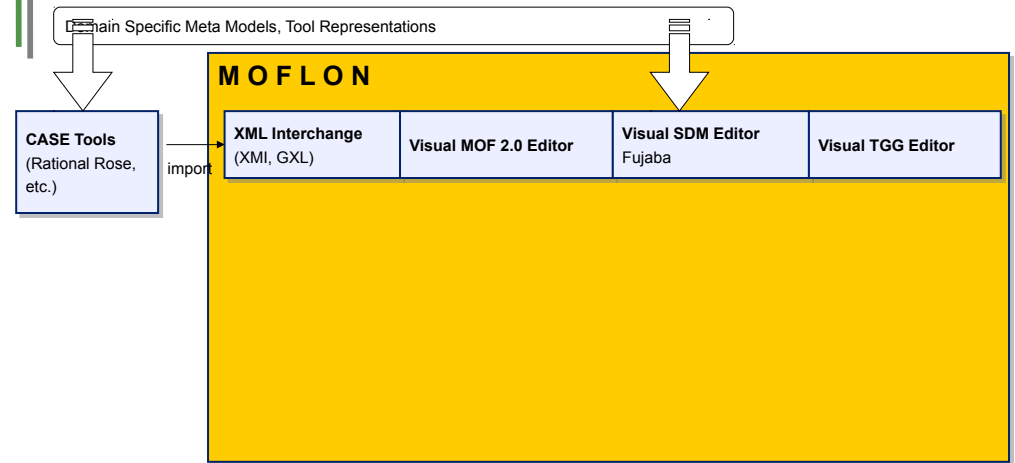
    final tudresden.oc120.core.lib.OclBoolean tud0c120Exp6 = (tudresden.oc120.core.lib.OclBoolean) tud0c120Eval0.forAll(
        tud0c120Iter1, tud0c120Exp1);

    return tud0c120Exp6;
}
    
```

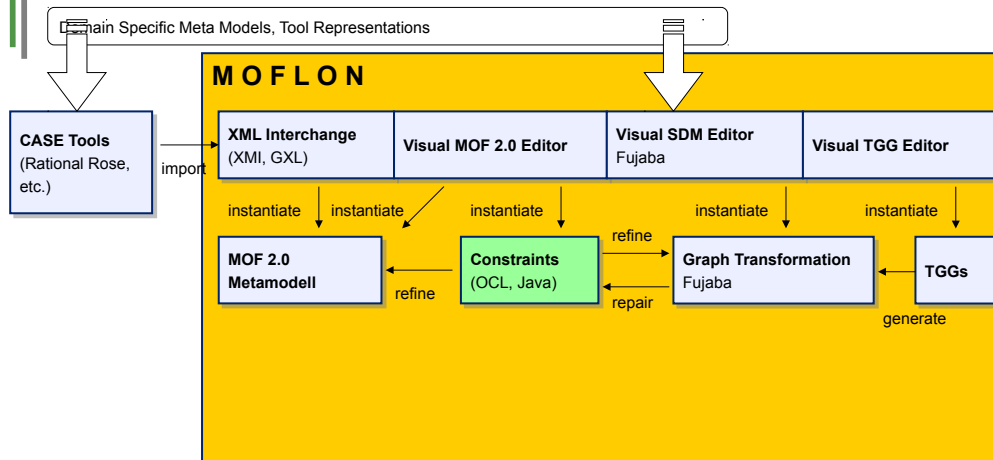
Result of MOFLON Example 1 – Statechart Editor (STaX)



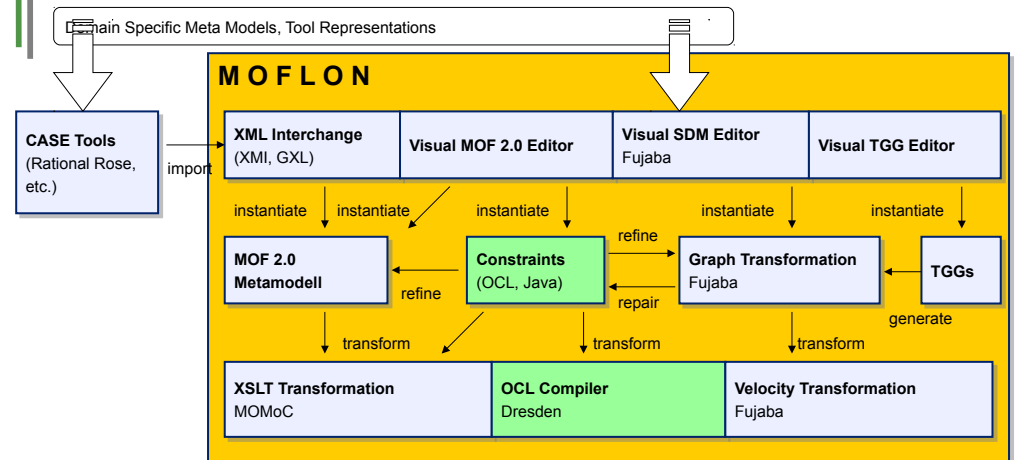
43.3.3 MOFLON – Architecture



MOFLON – Architecture



MOFLON – Architecture



TiE-CDDS – Constraints in Class Diagrams (2) Integration Framework

The screenshot shows the TiE Integration Framework interface. Two buttons at the top are labeled "load CD metamodel" and "load CD model". A "Constraint Validation" dialog box is open, displaying the following error message:

source domain 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: inv: attrs->size()>=1 and attrs->size()<=unbounded

An orange callout box points to the "model violates constraints:" section of the dialog, listing:

- 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

Another orange callout box points to the class diagram visualization in the LinkBrowser, stating: "visualization of classdiagrams model (here: source domain)".

TiE-CDDS – Constraints in Class Diagrams (3) Model Browser

The screenshot shows the TiE Integration Framework interface with the "JmiModelBrowser" window open. The browser displays a tree structure of classes and associations. A "String Editor Dialog" is open, showing a text input field with "surname" and "OK" and "Abbrechen" buttons. Below the dialog, a table shows the following data:

name	type	upper	lower
name	String	1	1
is_primary	Boolean	1	1
type	Classifier	-1	1

An orange callout box states: "model is fixed in generic model editor".

TiE-CDDS – Constraints in Class Diagrams (4) Integration Framework

The screenshot shows the TiE Integration Framework interface. A "Constraint Validation" dialog box is open, displaying the following message:

source domain model fulfills its constraints

An orange callout box states: "translation process may start now...".

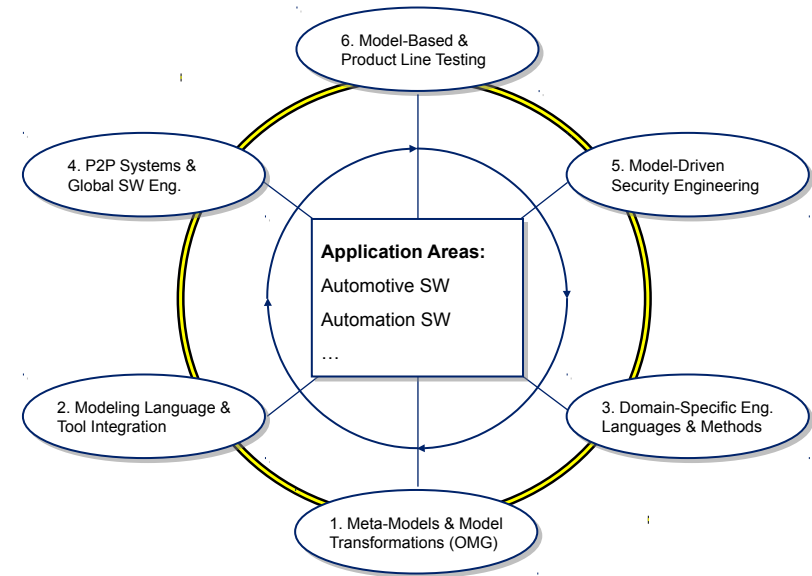
TiE-CDDS – Constraints in Class Diagrams (5) Forward Translation to DB representation

The screenshot shows the TiE Integration Framework interface with the "Forward Translation (Batch, Simple)" action selected. The LinkBrowser window displays a class diagram with green arrows indicating the translation process from the source domain to the target domain.

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	Fujaba & TGG	Progres & TGG	GME & GReAT	EMF & TeFkat	AToM ³	MetaEdit+	EMF & DSL	EMF & GMF	Pounamu	DiaGen	EBNF & TXL	SQL	XML	
Abstract syntax	+	+	+	+	+	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	o
Acceptability	+	+	o	o	o	+	--	+	--	o	+	o	o	+	+
Scaleability	+	+	--	o	--	o	--	o	--	o	--	--	--	--	o
Tool availability	o	o	o	+	+	+	+	+	o	o	+	+	+	+	o
Expressiveness	+	+	o	+	+	o	o	o	o	o	o	o	o	+	o

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 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. 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.

Some slides are courtesy Florian Heidenreich and Felix Klar

Thank you for your attention...

