

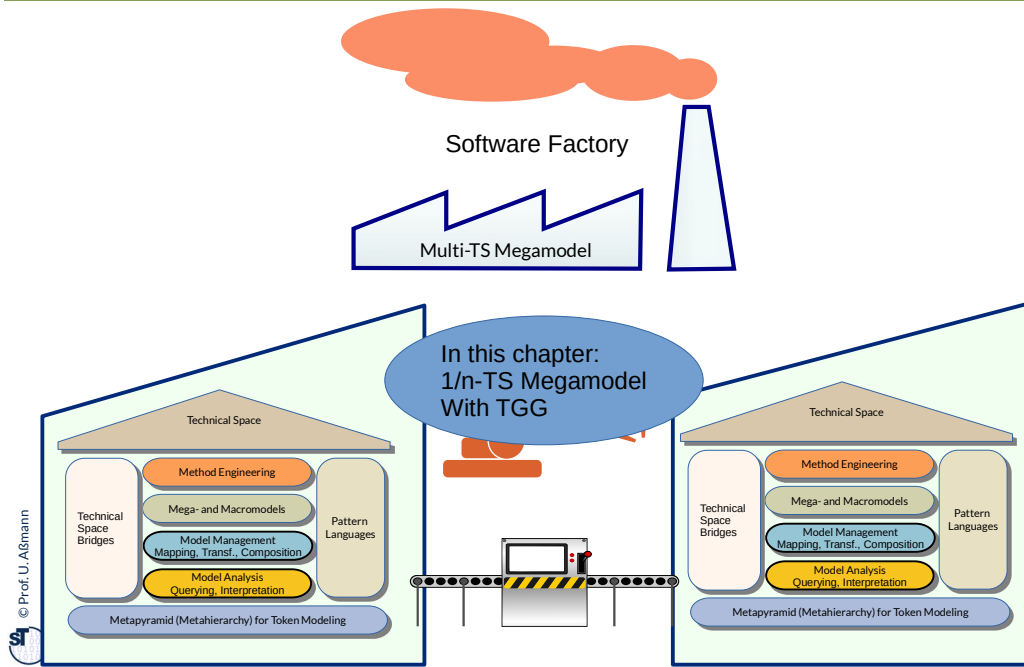


This slide set needs much more care and examples.
NOT, FORALL, etc.

- ▶ [ES89] Gregor Engels, Wilhelm Schäfer. Programming Environments, Concepts and Realization (in German), 1989, Teubner-Verlag Stuttgart
- ▶ Anthony Anjorin, Erhan Leblebici, and Andy Schürr. 20 years of triple graph grammars: A roadmap for future research. ECEASST, 73, 2015.
- ▶ F. Klar, A. Königs, A. Schürr: "Model Transformation in the Large", Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, New York: ACM Press, 2007; 285-294. <http://www.idt.mdh.se/esec-fse-2007/>
- ▶ www.fujaba.de www.moflon.org, <https://emoflon.org/>
 - <https://paper.dropbox.com/doc/Meta-Modelling-with-eMoflonCore--ArVO3r~geAdwkL9vVBUTzKZAg-zyOqELGZ0X9jL85TAs7pf>
- ▶ T. Fischer, J. Niere, L. Torunski, and A. Zündorf, 'Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language', in Proc. of the 6th International Workshop on Theory and Application of Graph Transformation (TAGT), Paderborn, Germany (G. Engels and G. Rozenberg, eds.), LNCS 1764, pp. 296--309, Springer Verlag, November 1998.
<http://www.upb.de/cs/ag-schaefer/Veroeffentlichungen/Quellen/Papers/1998/TAGT1998.pdf>

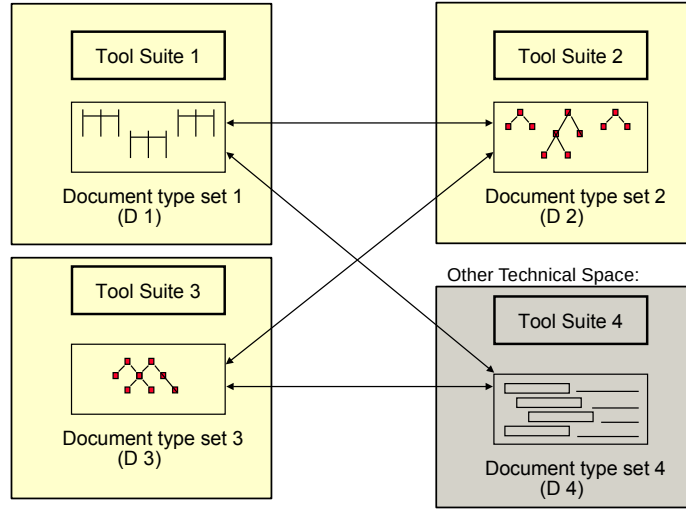
- ▶ [KS05] Alexander Königs, Andy Schürr. Multi-Domain Integration with MOF and extended Triple Graph Grammars. Technical Report. University of Technology Darmstadt. Dagstuhl Seminar Proceedings 04101
 - <http://drops.dagstuhl.de/opus/volltexte/2005/22>
- ▶ Alexander Königs, Andy Schürr. MDI: a rule-based multi-document and tool integration approach. *Softw Syst Model* (2006) 5:349–368 DOI 10.1007/s10270-006-0016-x
 - TGG between multiple documents and models
- ▶ [HJSWB] Florian Heidenreich, Jendrik Johannes, Mirko Seifert, Christian Wende and Marcel Böhme: Generating Safe Template Languages. In Proceedings of the "Eighth International Conference on Generative Programming and Component Engineering", GPCE'09, 4 - 5 October 2009, Denver, Colorado

Q13: A Software Factory's Heart: the Multi-TS Megamodel



Integration of Tool Suites by Data Connection

- ▶ Material of several tool (suites) can be **data-connected** by transformations or access adaptations



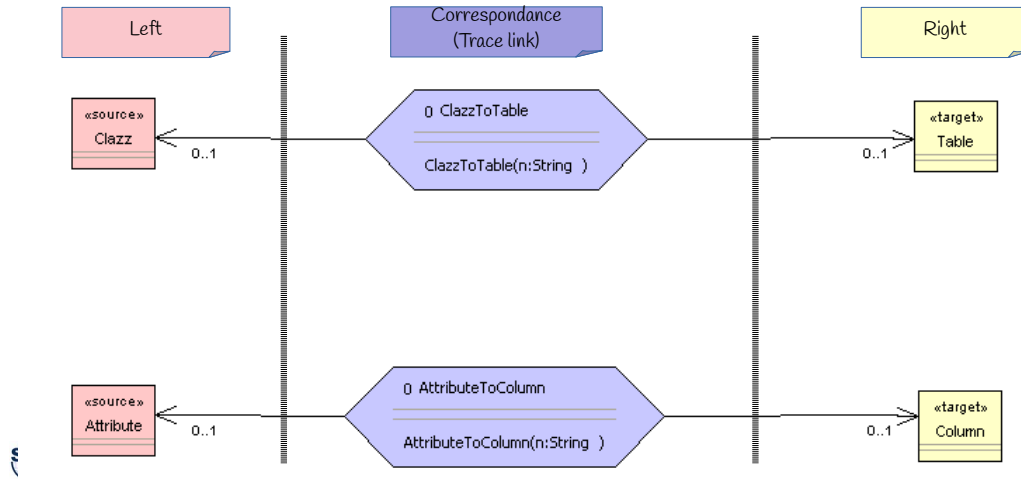


54.1 „Synchronizing“ Models with Triple Graph Grammars

- Mapping graphs to other graphs, also in data connections of different tools
- Specification of mappings with mapping rules
- Incremental transformation
- Traceability

Triple Graph Grammars – Moflon Example

- ▶ A **Triple Graph Grammar (TGG)** is a mapping-oriented transformation system, consisting of rules with three „areas“ (better called **metamodel mapping grammars**)
 - Left side: (source) graph pattern 1 in (source) graph 1
 - Right side: (target) graph pattern 2 in (target) graph 2
 - **Middle: relational expression (net)** relating graph pattern 1 and 2 (trace model)



Basic Types of Synchronization Rules

Depending on the modification colors, a TGG rule can be checking or creating the correspondance.

Rule classes from [KS05] Koenigs/Schuerr 2005:

- ▶ **Consistency Checking rules** – test whether both patterns exist
 - modification color is black (test)
- ▶ **Traceability relationship creating rule** – add a trace relation between elements of both sides
 - modification color is green in correspondance part (add)
- ▶ **Create model element** in one domain matching its correspondant
 - modification color is green on one side (add)
- ▶ **Lower layer create model element** – create model in a lower grammar layer
 - modification color is green on lower layer (add)

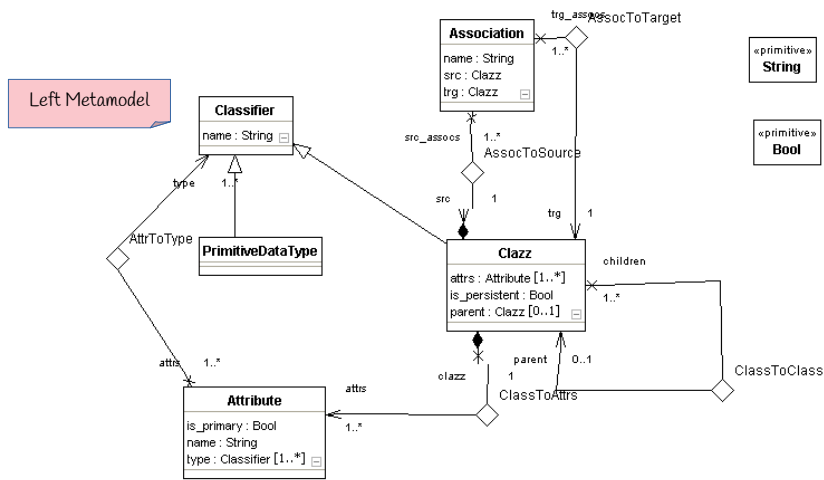


54.2.1. Mapping Objects to Tables (Object-Relational Mapping, ORM)

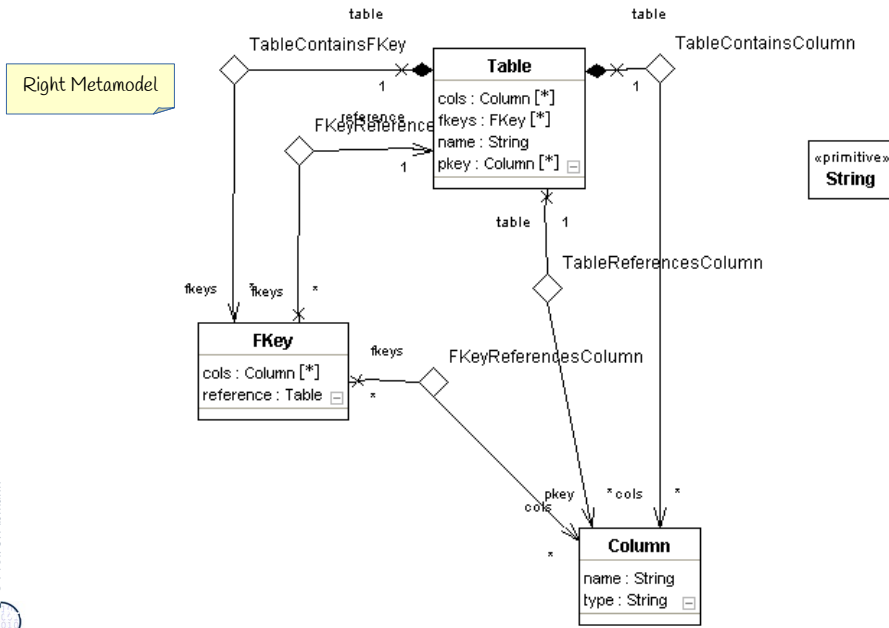
TGG for Object-Relational Mapping (ORM)

Left Metamodel: Class Diagram Metamodel (CD)

- ▶ Synchronize Class-Diagram-metamodel (CD) with a relational schema (RS): object-relational mapping (ORM)

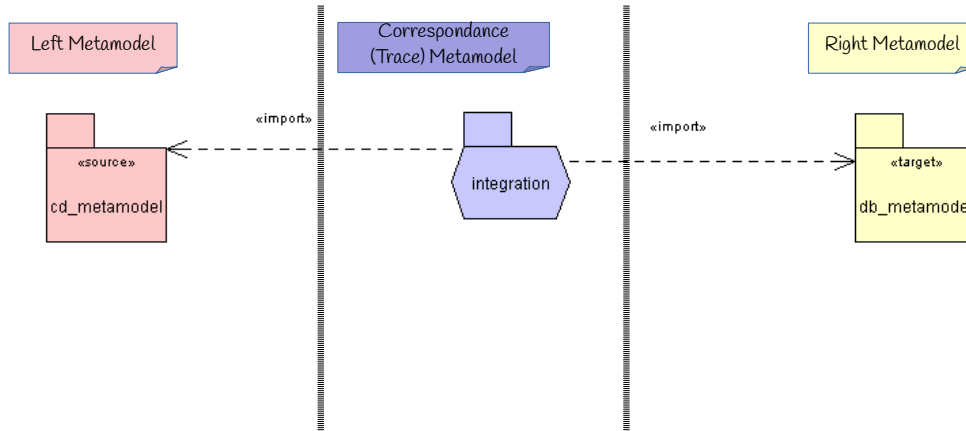


Right Metamodel: Relational Metamodel (DB, relational schema)



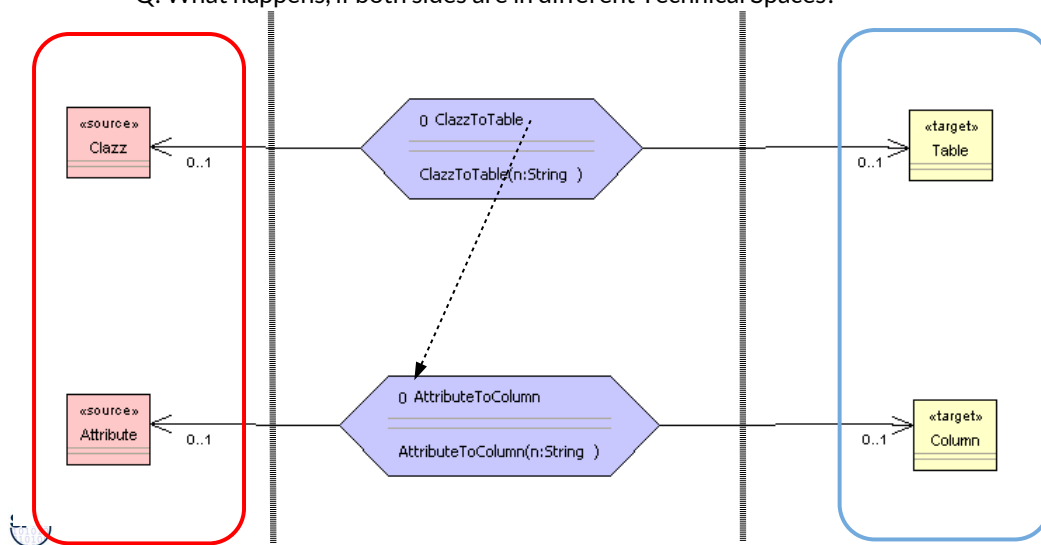
TGG for Object-Relational Mapping (ORM)

- ▶ The metamodel mapping grammar of a TGG has a top rule (start rule) which describes the relationship of the graphs on topmost level



Example of Consistency-Checking Rule

- ▶ From the top-rule `ClazzToTable`, other TGG rules are associated („called“/“invoked“)
- ▶ In this case, the TGG only checks (black color - TEST)
- ▶ Q: What happens, if both sides are in different Technical Spaces?



TGG Specify Transformation Bridges Between Roles and Technical Spaces

- ▶ TGG can also be used to data-exchange and synchronize Material classes and roles
 - between two material objects
 - between two tools with different repositories
 - even in different technical spaces
- ▶ The only assumption: 1:1 mappings of model elements

TGG are a fine technique to build *transformation bridges for data connection* between tools, even in different technical spaces.



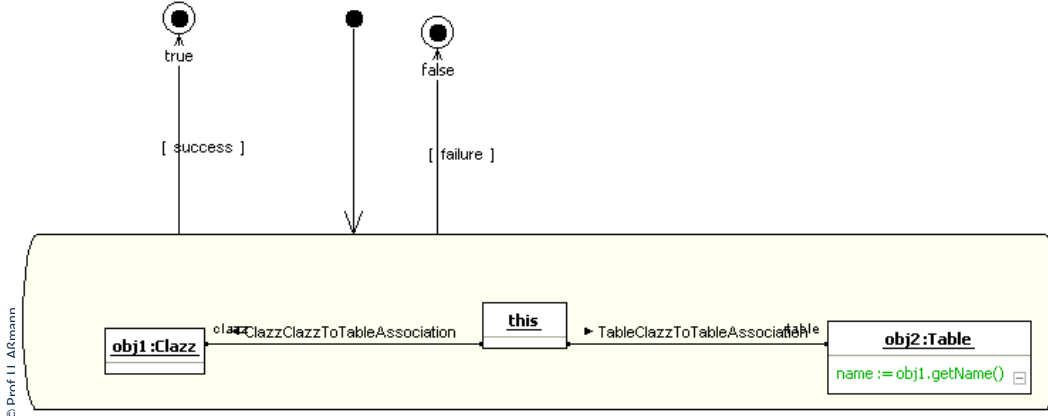
54.2. Triple Graph Grammars in MOFLON

- MOFLON in MOF Technical Space
- eMOFLON in EMOF TS

Triple Graph Grammars – Moflon Example

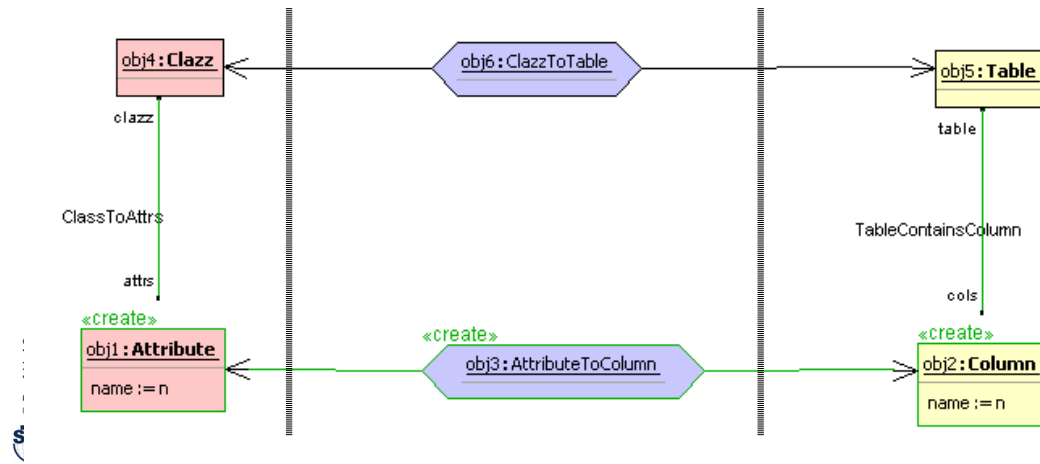
- ▶ Because they are named, TGG rules can be started by Fujaba Storyboards (activity diagrams)
- ▶ The activities can be associated to a transformation class `ClazzToTable`

`ClazzToTable::performForwardAttributeValuePropagation ()`: Boolean



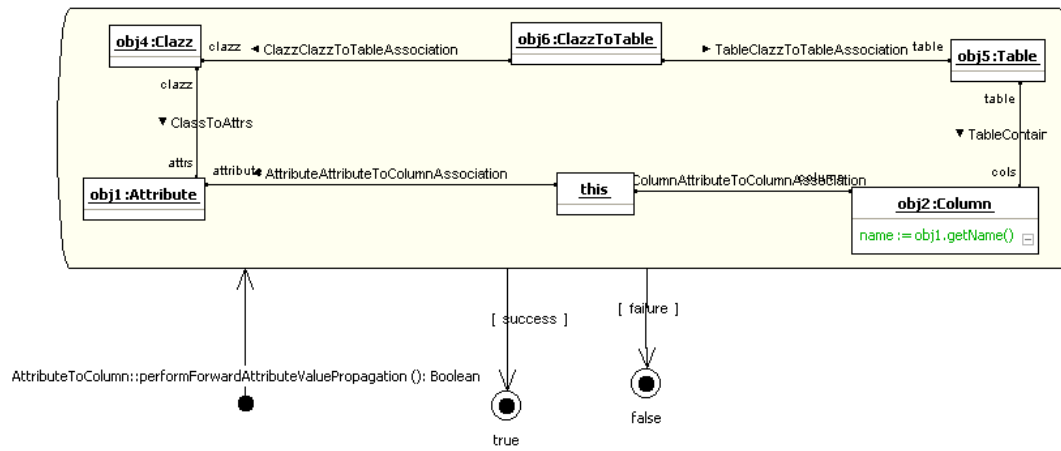
Example of Lower-Level-Creation Rule

- ▶ *Lower-level-creation rule* creates lower level elements and a pairwise correspondence of model elements on both sides
 - Here, objects on the lower level are created anew if needed from the tested upper level



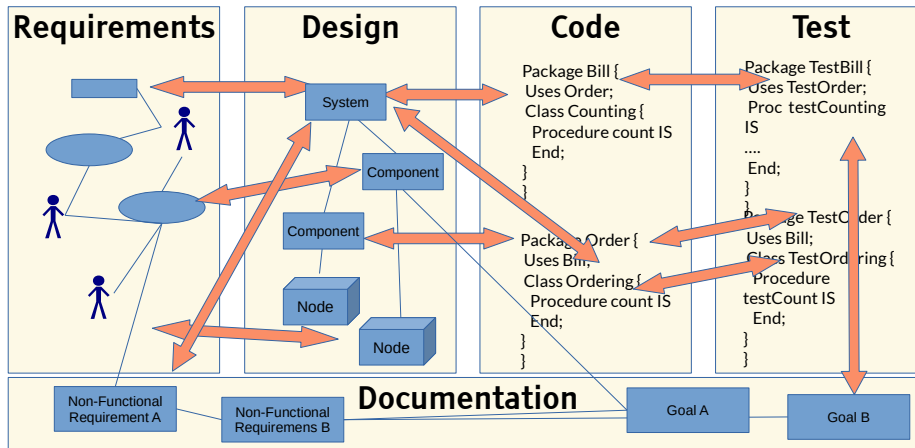
Triple Graph Grammars – Moflon Example

- ▶ Notation in Moflon/Fujaba Storyboards
- ▶ Checking a pattern with adding an attribute to obj2

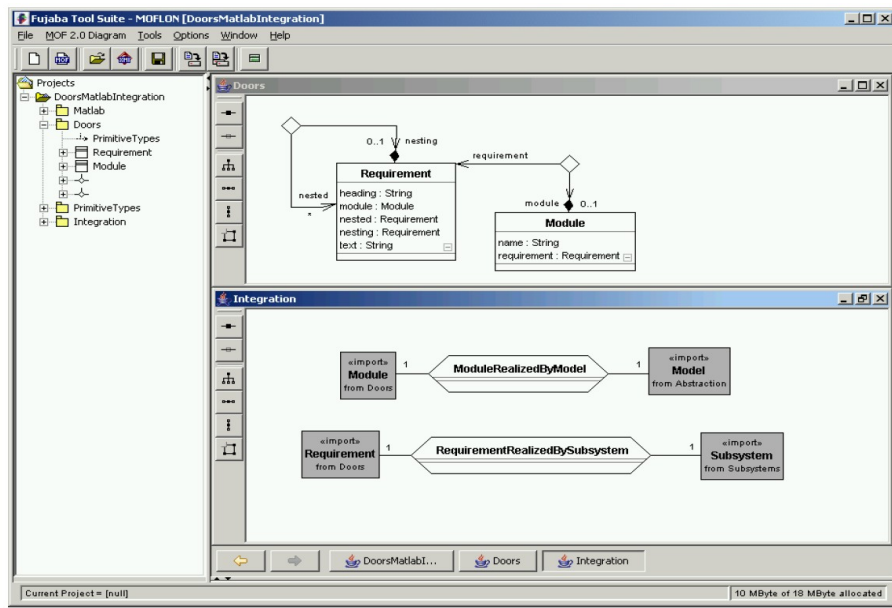


Q12: The ReDoDeCT Problem and its Macromodel

- ▶ The **ReDoDeCT problem** is the problem how requirements, documentation, design, code, and tests are related (\rightarrow V model)
- ▶ Mappings between the Requirements model, Documentation files, Design model, Code, Test cases
- ▶ A **ReDoDeCT macromodel** has maintained mappings between all 5 models



Ex. 2: TGG Coupling of Requirements Specification and Design

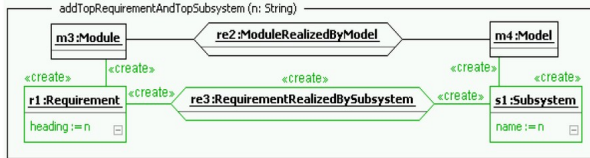


TGG Coupling Requirements Specification and Design

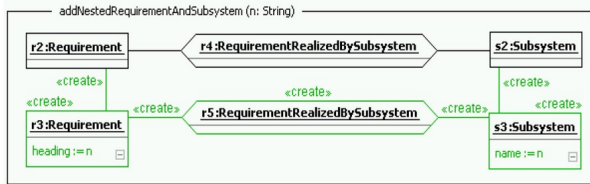
- ▶ This TGG grammar builds up a module-requirements graph
- ▶ Starting from a relation “ModuleRealizedByModel” and “RequirementRealizedBySubsystem”



initial, creational



lower-layer
creational

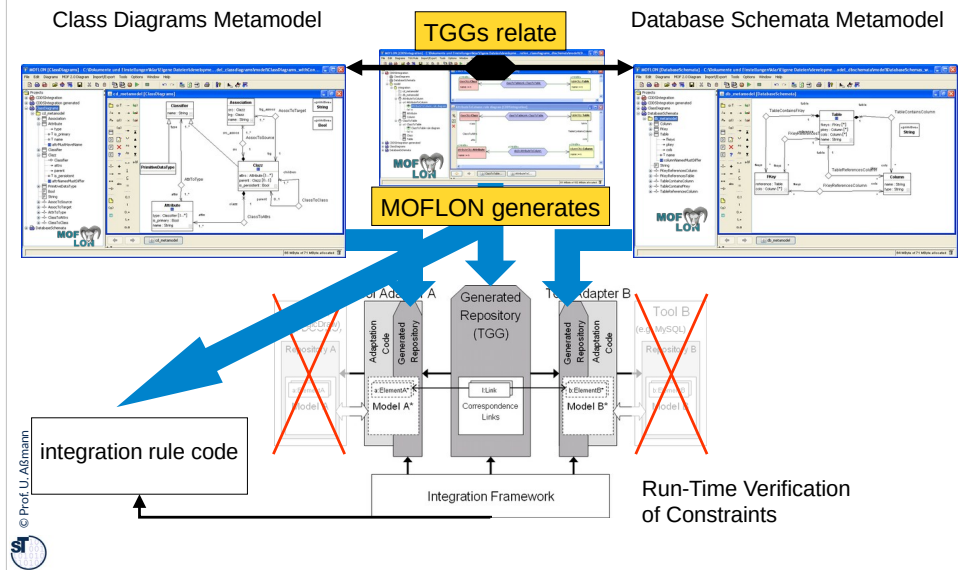


lower-layer
creational



54.3. Using Triple Graph Grammars in MOFLON

Example: Object-Relational Mapping "TIE-CD-DB": (ClassDiagrams / DatabaseSchema)



shows how our architecture is realized with the MOFLON metamodeling and translation specification approach

TiE-CD-DB - Constraints in Class Diagrams (1) Generate Code from MOF model (CD metamodel)

The image displays three overlapping windows from the MOFLON software:

- Top Left Window:** Shows a class diagram metamodel with classes like `Classifier`, `Association`, `Classz`, and `Attribute`. A red circle highlights the `Classz` class.
- Bottom Left Window:** An "Edit MOF Constraint" dialog box. The "General" tab is active, showing the constraint name `attrNamesMustDiffer`, the language `OCL`, and the body: `inv:attrs->forall(a1, a2:Attribute) a1 <> a2 implies a1.name <> a2.name`. The "Visibility" section has `public` selected.
- Right Window:** A context menu for the `cd_metamodel` package. The "Generate MOFLON-Code" option is highlighted with a red rectangle.



TiE-CD-DB - Constraints in Class Diagrams (2)

Loading Metamodels and Models

The screenshot displays the TiE Integration Framework interface. At the top, there are two buttons: "load CD metamodel" and "load CD model". Below these, a table lists tool adapters for Source, Target, and Link domains. A "Constraint Validation" dialog box is open, displaying error messages. At the bottom, a class diagram model is visualized, showing a hierarchy of classes and their attributes.

load CD metamodel

load CD model

Constraint Validation

source domain model does not fulfill its constraints:
constraint named 'attrNamesMustDiffer' is violated in instance: Customer: inv:attrs->ForAll(a1: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.Classz': should be [1,unbounded] but is 0: inv: attrs->size()>=1 and attrs->size()<=unbounded

model violates constraints:

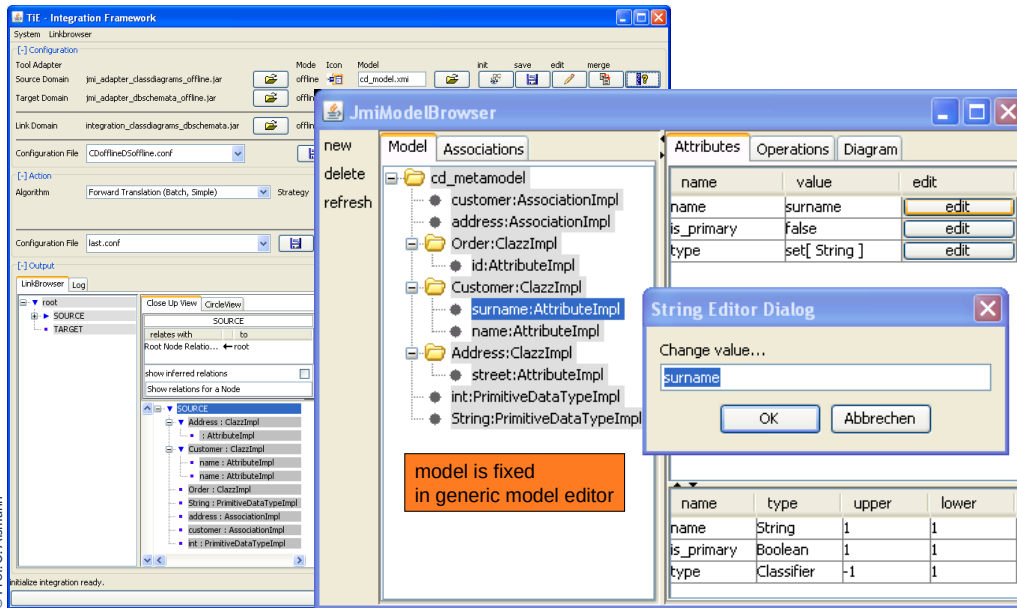
- class „Customer“ has two attributes with same name: „name“
- attribute in class „Address“ has no name
- multiplicity violation: class „Order“ has no attribute but according to CD metamodel every class must have one

visualization of classdiagrams model (here: source domain)

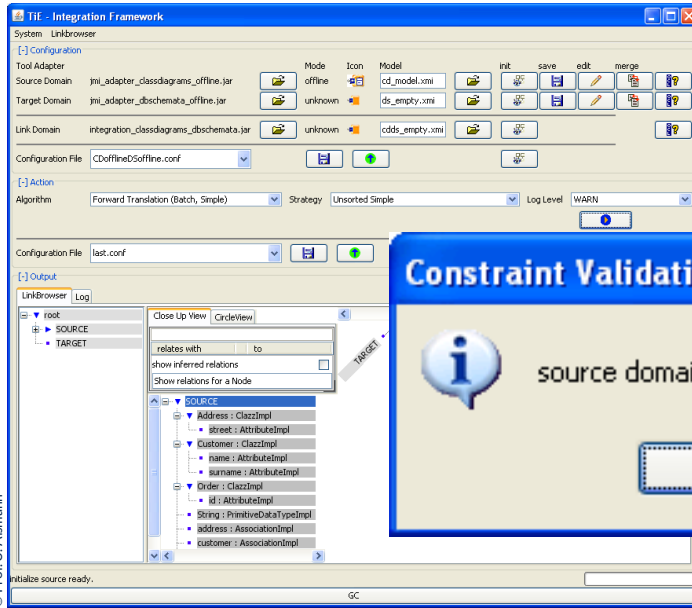
© Prof. U. A. Gmann

TiE-CD-DB - Constraints in Class Diagrams (3)

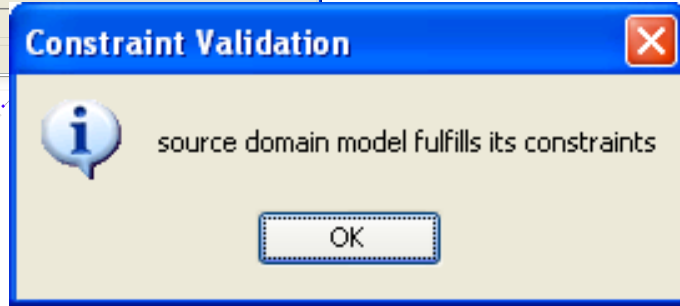
Model Browser



TiE-CD-DB - Constraints in Class Diagrams (4) Integration Framework



translation process
may start now...



TiE-CD-DB - Constraints in Class Diagrams (5)

Forward Translation to DB representation

The image displays two side-by-side screenshots of the TiE - Integration Framework application. Both windows show the 'System Linkbrowser' and 'Configuration' sections. The left window is set to 'Forward Translation (Batch, Simple)' with configuration file 'last.conf'. The right window shows the 'Output' section with a 'LinkBrowser' and a 'Close Up View' of a class diagram. The diagram shows a 'SOURCE' domain with classes like 'Address :ClazzImpl', 'Customer :ClazzImpl', and 'Order :ClazzImpl', and a 'TARGET' domain with classes like 'String :PrimitiveDataTypeImpl' and 'Integer :PrimitiveDataTypeImpl'. Green arrows indicate the mapping from source classes to target classes. A 'Target' label with an arrow points to the right window.



Other Software Engineering Applications of Model Synchronization

- ▶ Mapping a PIM to a PSM in Model-Driven Architecture
- ▶ Graph Structurings (see course ST-II)
- ▶ Refactorings (see Course DPF)
- ▶ Semantic refinements
- ▶ Round-Trip Engineering (RTE)

54.4 The Tornado Method: Specification of TGG Rules using Textual Concrete Syntax

- Slides about Tornado courtesy to Mirko Seifert and Christian Werner
- Presented at Fujaba Days 2009, Eindhoven, The Netherlands, 16.11.2009
- Christian Werner. Konzeption und Implementierung eines Debuggers für textuelle Triple Graph Grammar Regeln. Belegarbeit, Lehrstuhl Softwaretechnologie, 2010, TU Dresden
- available on request

Motivation for Textual Syntax of TGG

- ▶ TGGs are fine for model synchronization, but writing TGG rules is not always easy

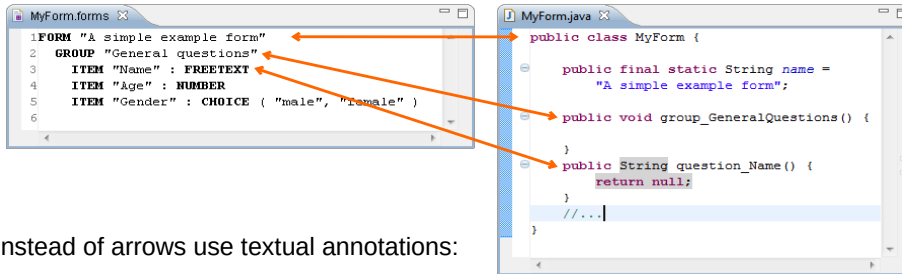
Why?

- ▶ Rule specification typically on the level of abstract syntax
 - Complex abstract syntax (AS) graphs vs. simple concrete syntax (CS) fragments
 - Rule designers not always familiar with AS
- ▶ Rule specification is based on graphical syntax
 - But: There is lots of textual (modelling) languages
 - Gap: Graphical rules vs. textual models
 - Large graphical rules are hard to read

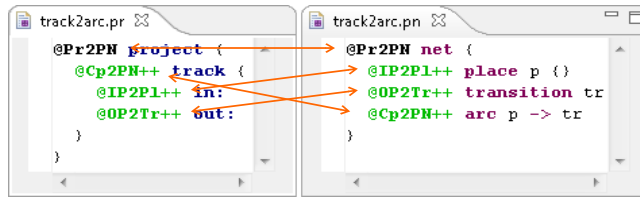
Can we do better?

Idea for Rule Specification in EMFText

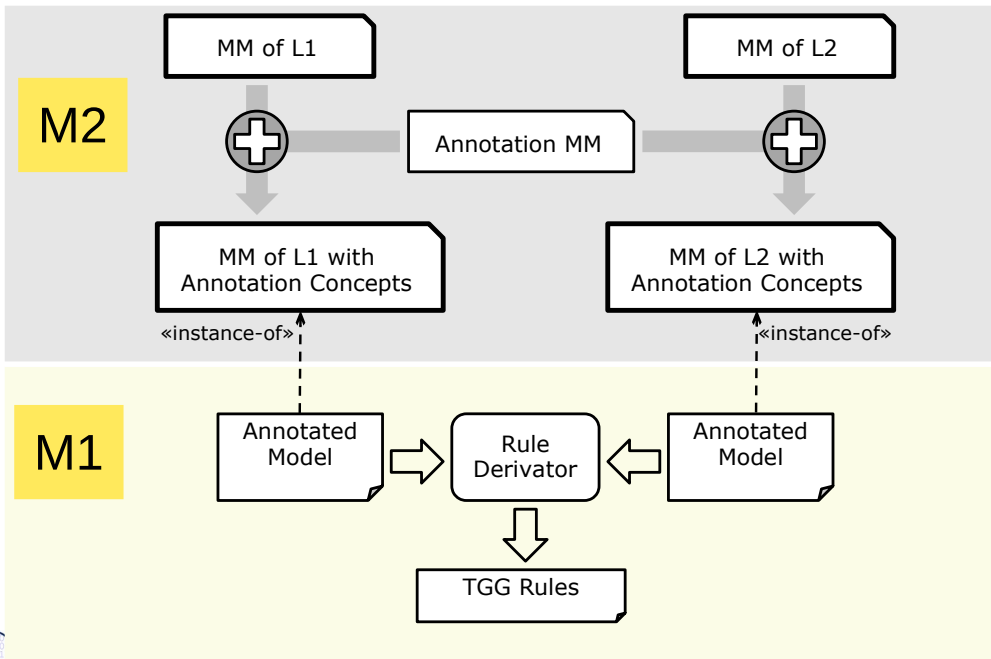
- ▶ employ EMFText; use concrete textual syntax of involved languages
- ▶ derive rules from pairs of models
- ▶ do it in a generic way (automatic application to any language)



Instead of arrows use textual annotations:

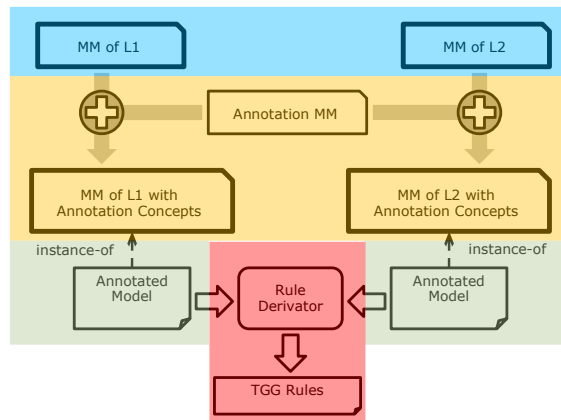


Tornado Generation Process of TGG Rules



Generation Steps of Tornado Method

1. Make meta models extensible
2. Extend meta models (with annotation concepts)
3. Extend concrete syntax
4. Derive rules from model pairs



Step 1 – Getting (more) Extensible Metamodels

Extensibility provided by Ecore (EMOF):

- ▶ Add new metaclasses (i.e., new complex types)
- ▶ Reference existing metaclasses (Reuse)
- ▶ Subclass existing metaclasses

What is missing in EMOF:

- ▶ Distinction between subtyping and inheritance
- ▶ Extensibility for primitive types
- ▶ Example:
 - Can't add things that do not have a property year
 - Can't add subtypes for EString



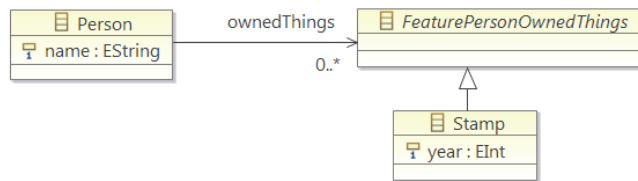
Step 1a - Getting Extensible Metamodels

Separate subtyping and inheritance (algorithm from [HJSWB]):

For each feature's type that has at least one superclass or defines at least one feature:

- ▶ Introduce a new abstract metaclass `Feature<ClassName><FeatureName>`
- ▶ Change the type of the feature to the new metaclass
- ▶ Make the former type of the feature a subclass of the new metaclass

Example:



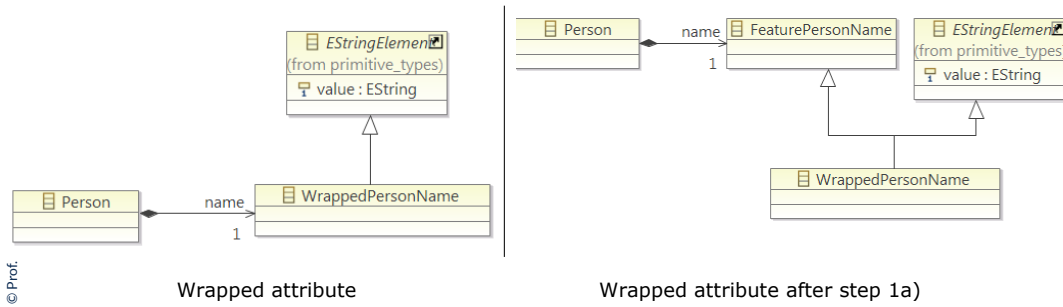
Step 1b - Getting Extensible Metamodels

Wrap primitive types (also from [HJSWB]):

For each attribute that has a primitive type:

- ▶ Create a new subclass of the primitive type wrapper class
- ▶ Replace attribute with reference to new subclass

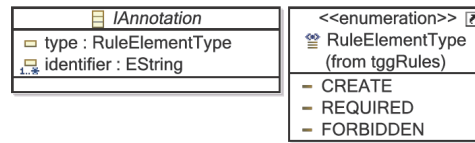
Example:



Step 2 – Extending Metamodels with Annotation Concepts

Goal:

- ▶ Every model element can be annotated



HowTo:

- ▶ For each meta class `X` create new metaclass `AnnotableX` with
 - Reference to class `Annotation` (to store the annotation)
 - Reference to the original class `X` (to store the data of `X`)
 - Make `AnnotableX` a subclass of each feature class that `X` inherits from (to make `AnnotableX` usable wherever `X` can be used)

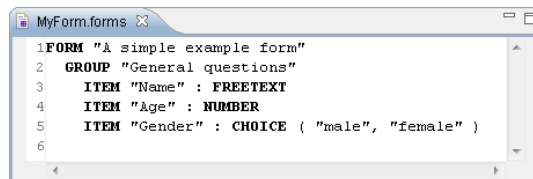
Step 3 – Extending the Concrete Syntax Specification

- ▶ Steps 1 and 2 added annotations concepts on the level of abstract syntax, but concrete one is need to write them down
- ▶ Textual syntax tools (e.g., EMFText, xText and TCS) use one rule per metaclass
 - Retain the existing syntax rules
 - Add syntax rules for new annotation classes in meta model

```
Form ::= "FORM" name['', ''] groups*;  
AnnotableForm ::= (identifier[IDENT])+ (type[TYPE])? form;
```

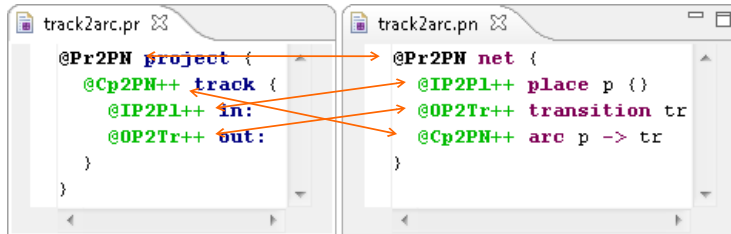
@1 ! FORM "A simple example form"

IDENT is some identifier starting
with an @
TYPE is !, ++ or --



Step 3 – Extended Concrete Syntax

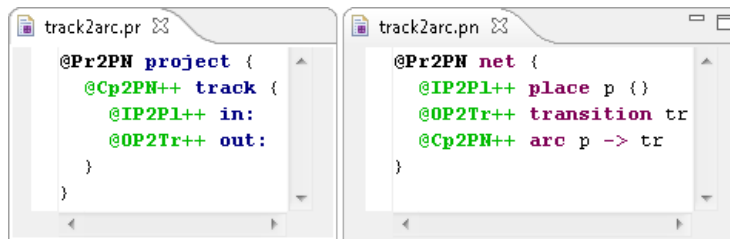
Rail tracks to Petrinet example



(bold black and green elements are new – TGG rule annotations)

Step 4 – Deriving Rules from Model Pairs

- ▶ For each annotated model element, create a rule node
- ▶ For each set of model elements that are annotated with the same identifier,
 - create a correspondence node and create links connecting the new correspondence node with the respective rule nodes
- ▶ Mark all rule nodes as “create” where the corresponding model element is annotated as create element
- ▶ For each pair of model elements that is connected by exactly one reference
 - create a link between the respective rule nodes
- ▶ For each pair of model elements that is connected by multiple references
 - use the references specified in the annotation
 - and create links between the respective rule nodes



The image shows two side-by-side code editor windows. The left window, titled 'track2arc.pr', contains the following code:

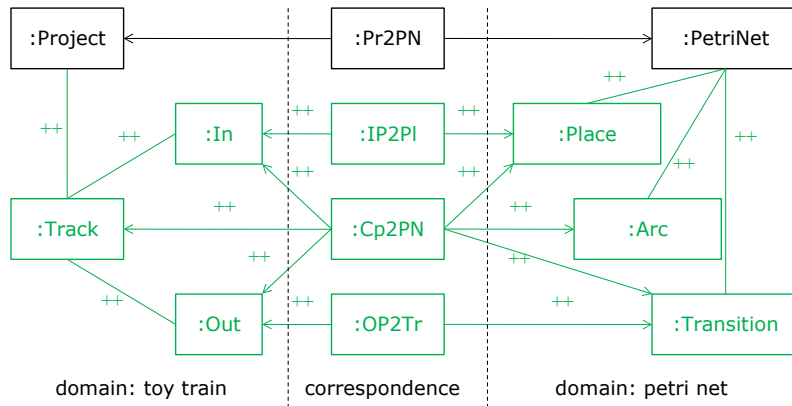
```
@Pr2PN project {
  @Cp2PN++ track {
    @IP2P1++ in:
    @OP2Tr++ out:
  }
}
```

The right window, titled 'track2arc.pn', contains the following code:

```
@Pr2PN net {
  @IP2P1++ place p {}
  @OP2Tr++ transition tr
  @Cp2PN++ arc p -> tr
}
```

Step 4 – Deriving Rules from Model Pairs

Rail tracks to Petri net example



Restrictions of Tornado

Constraints

- ▶ Can be derived (e.g., equality if attribute values match), but:
 - What about boolean attributes?
 - What about more complex constraints (a.name == b.id)?

Negative Application Conditions

- ▶ May need additional annotations

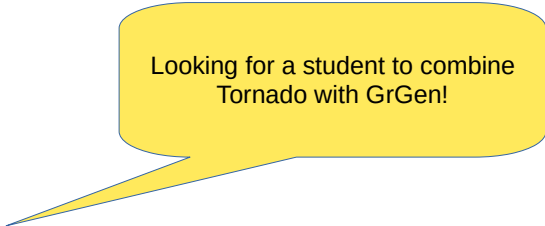
Concrete Syntax (CS) restricts rules that can be specified

- ▶ If AS is less restrictive than CS (e.g., metaclasses with empty CS)

Conclusion of (Experimental) Tornado Method

Textual (modelling) languages can be automatically extended with:

- annotation support (This whole stuff is for free!)
- other features (More stuff is for free as well! See e.g. [1])
- ▶ Rule specification using concrete syntax seems intuitive
- ▶ Combines benefits of specification by example (CS) and classic rule specification (precision)
- ▶ Debugging based on CS is enabled
- ▶ More annotations may be needed, but can easily be added
- ▶ Metamodelling languages should support extensibility to its full extent



Looking for a student to combine
Tornado with GrGen!

The End: What Have We Learned

- ▶ Graph rewrite systems are tools to transform graph-based models and graph-based program representations
- ▶ MOFLON supports OCL queries and constraints
- ▶ TGG enable to bidirectionally map models and synchronize them
- ▶ Why can a TGG also be called a *metamodel mapping grammar*?
- ▶ Correspondances in models can be expressed by annotations