

54. How to Synchronize Models with Triple Graph Grammars for Data Connection

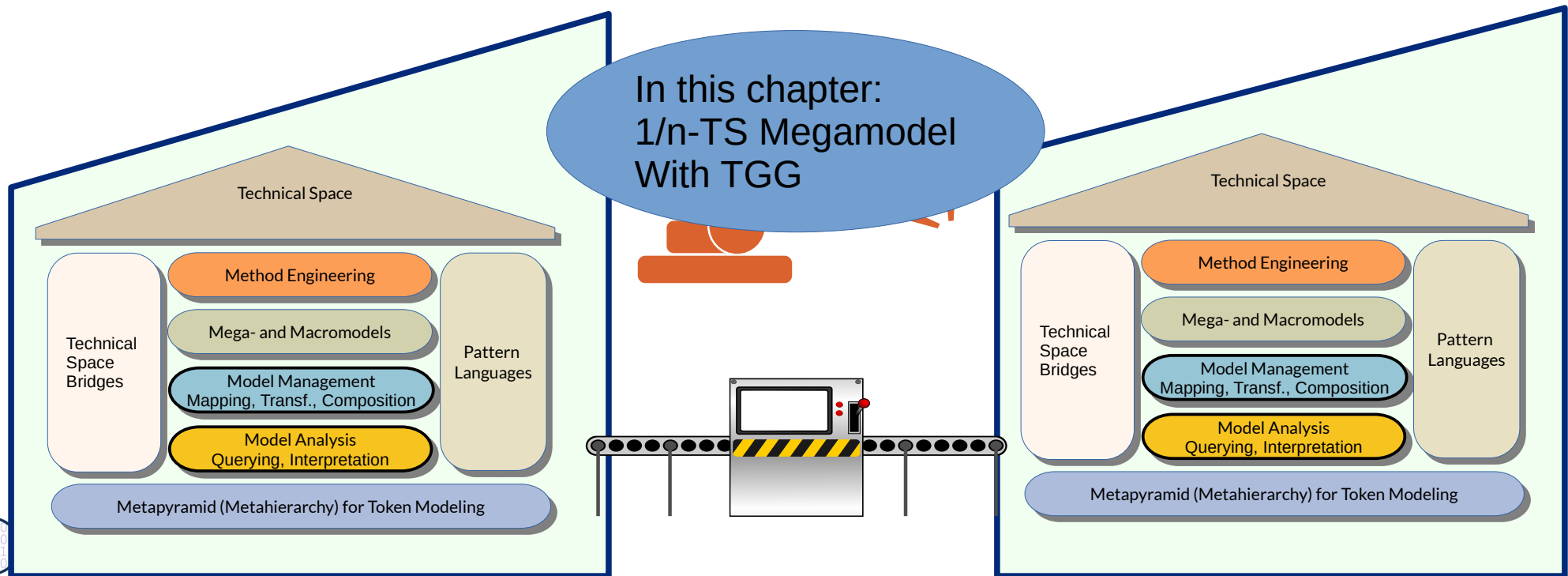
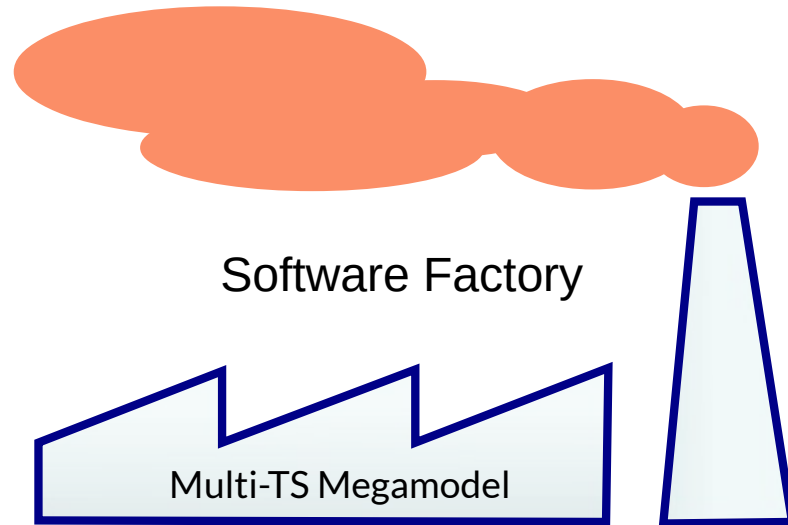
Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und
Multimediatechnik
Gruppe Softwaretechnologie
<http://st.inf.tu-dresden.de>
Version 21-0.5, 29.01.22

- 1) Triple Graph Grammars
- 2) Specifying TGG in MOFLON
- 3) Using TGG in MOFLON
- 4) The Tornado mapping method

- ▶ [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>

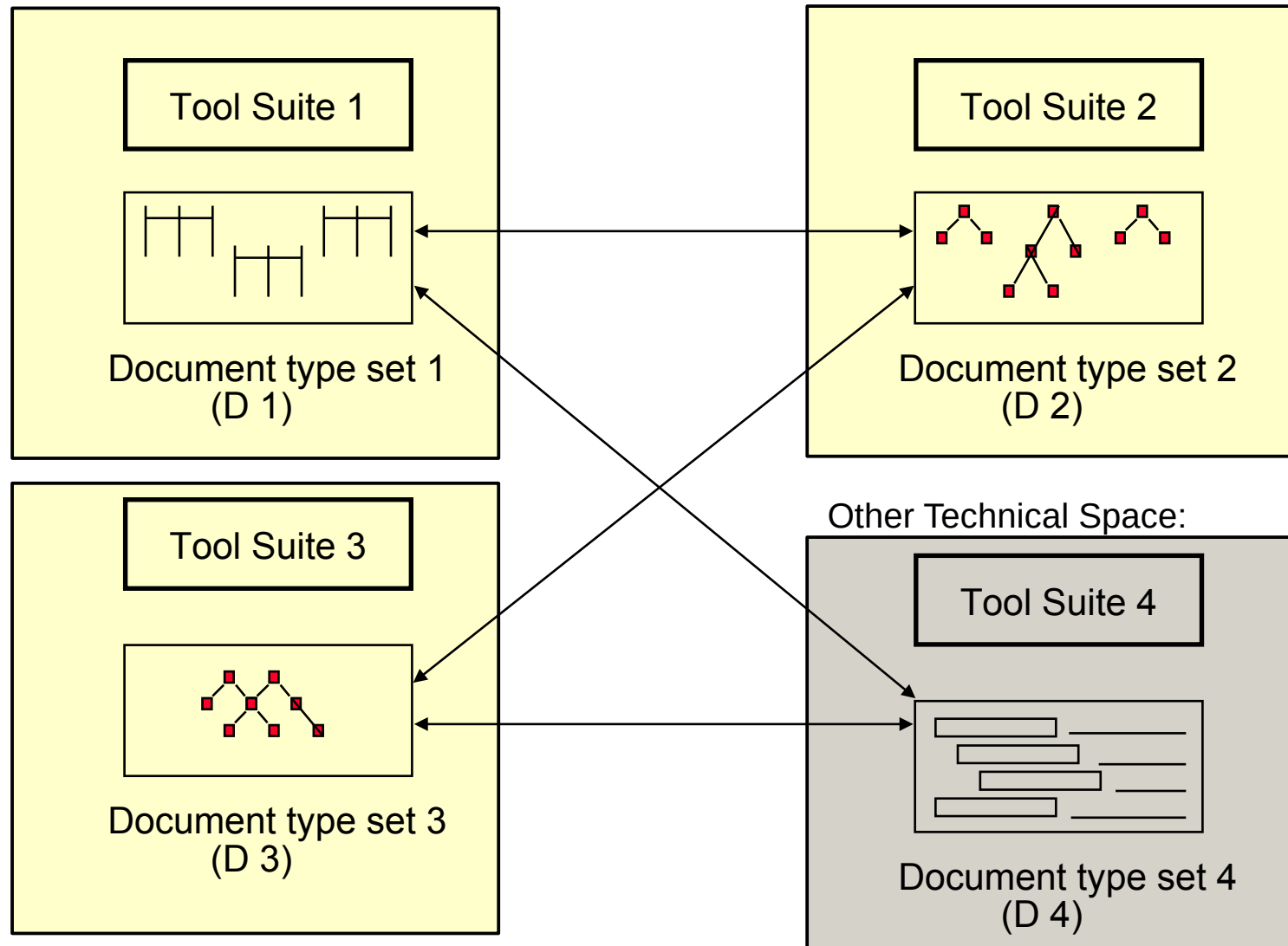
- ▶ [KS05] Alexander Königs, Andy Schürr. Multi-Domain Integration with MOF and extended Triple Graph Grammars. Technical Report. University of Technology Darmstadt. Dagstuhl Seminar Proceedings 04101
 - <http://drops.dagstuhl.de/opus/volltexte/2005/22>
- ▶ Alexander Königs, Andy Schürr. MDI: a rule-based 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



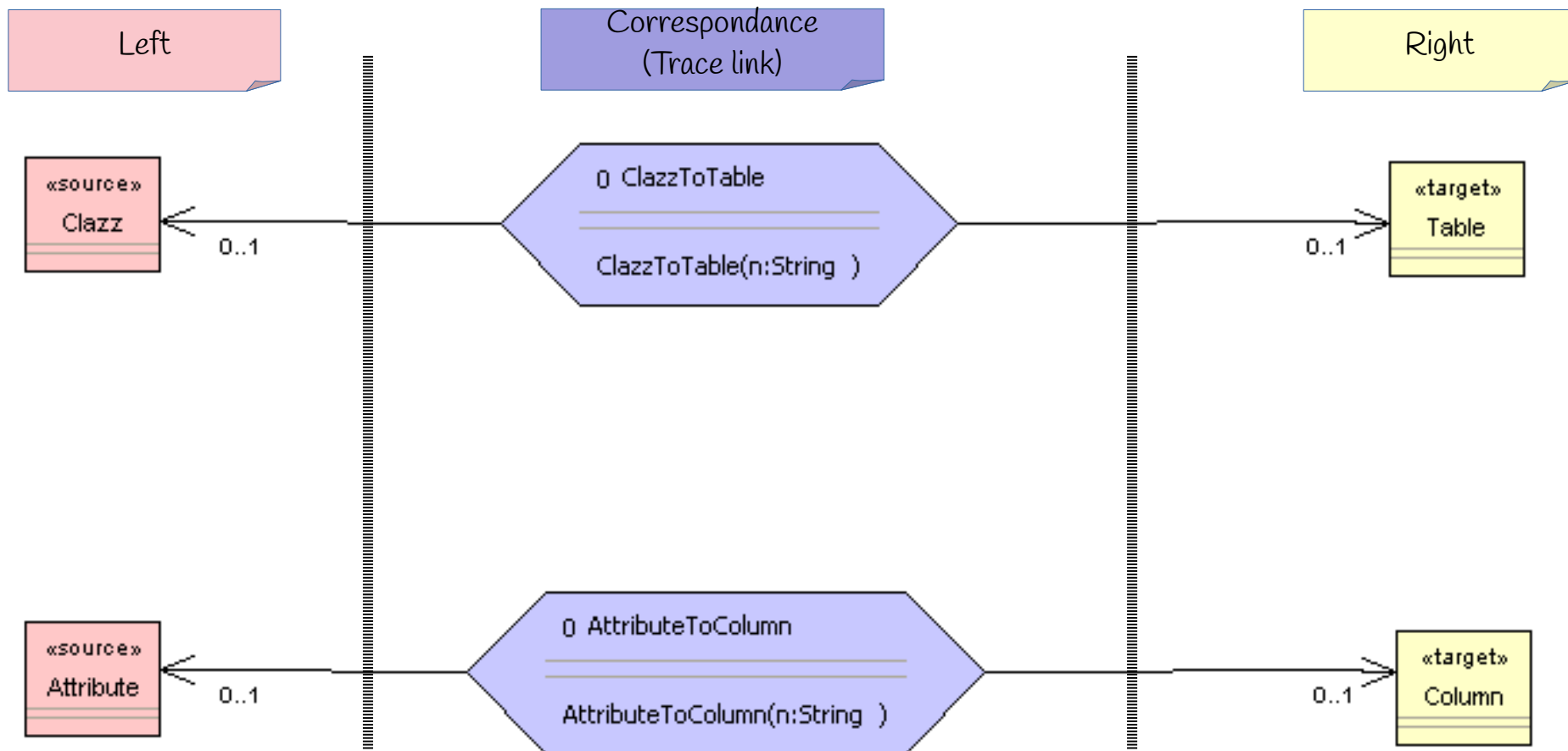


54.1 „Synchronizing“ Models with Triple Graph Grammars

- 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)

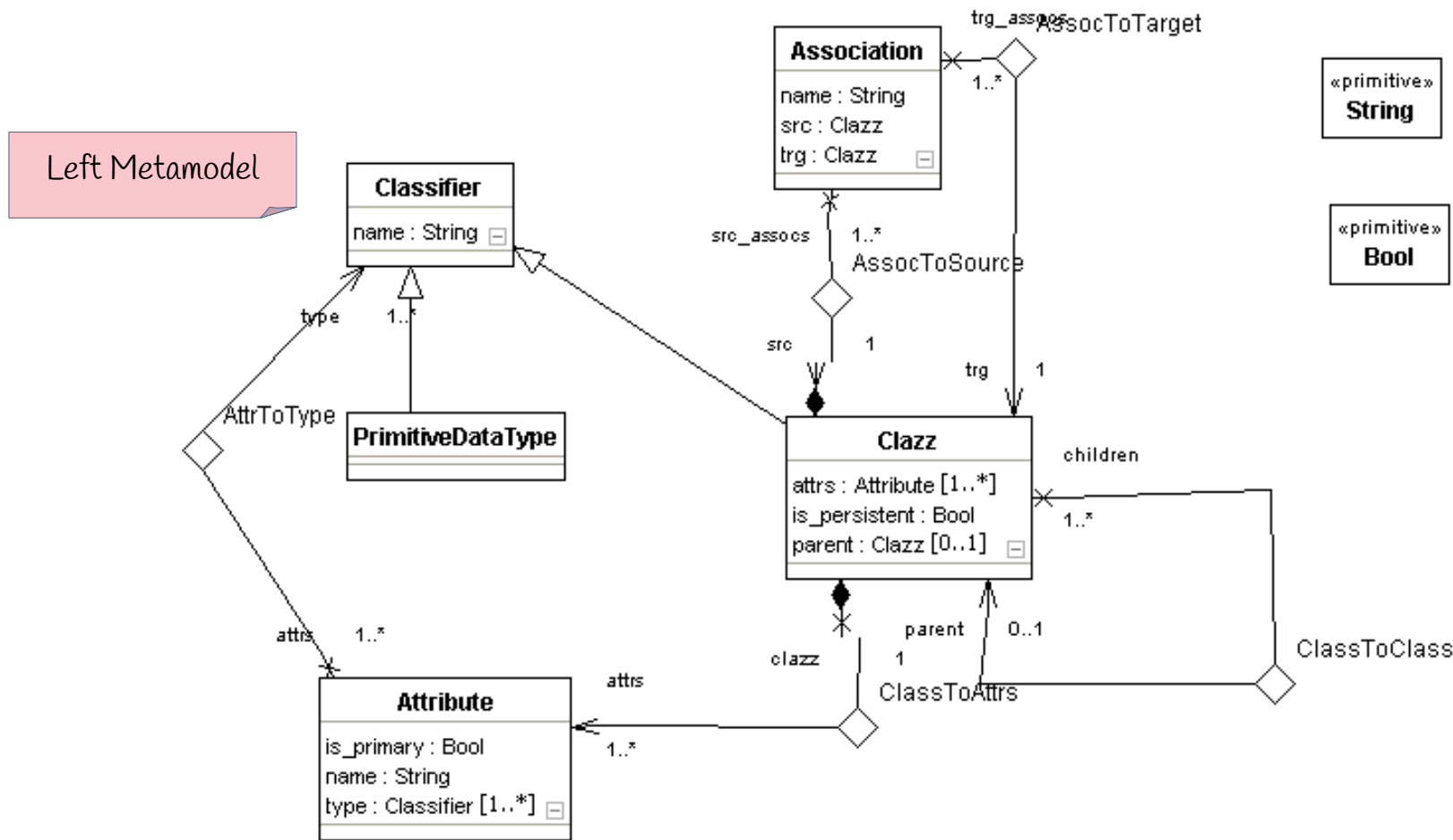


54.2.1. Mapping Objects to Tables (Object-Relational Mapping, ORM)

TGG for Object-Relational Mapping (ORM)

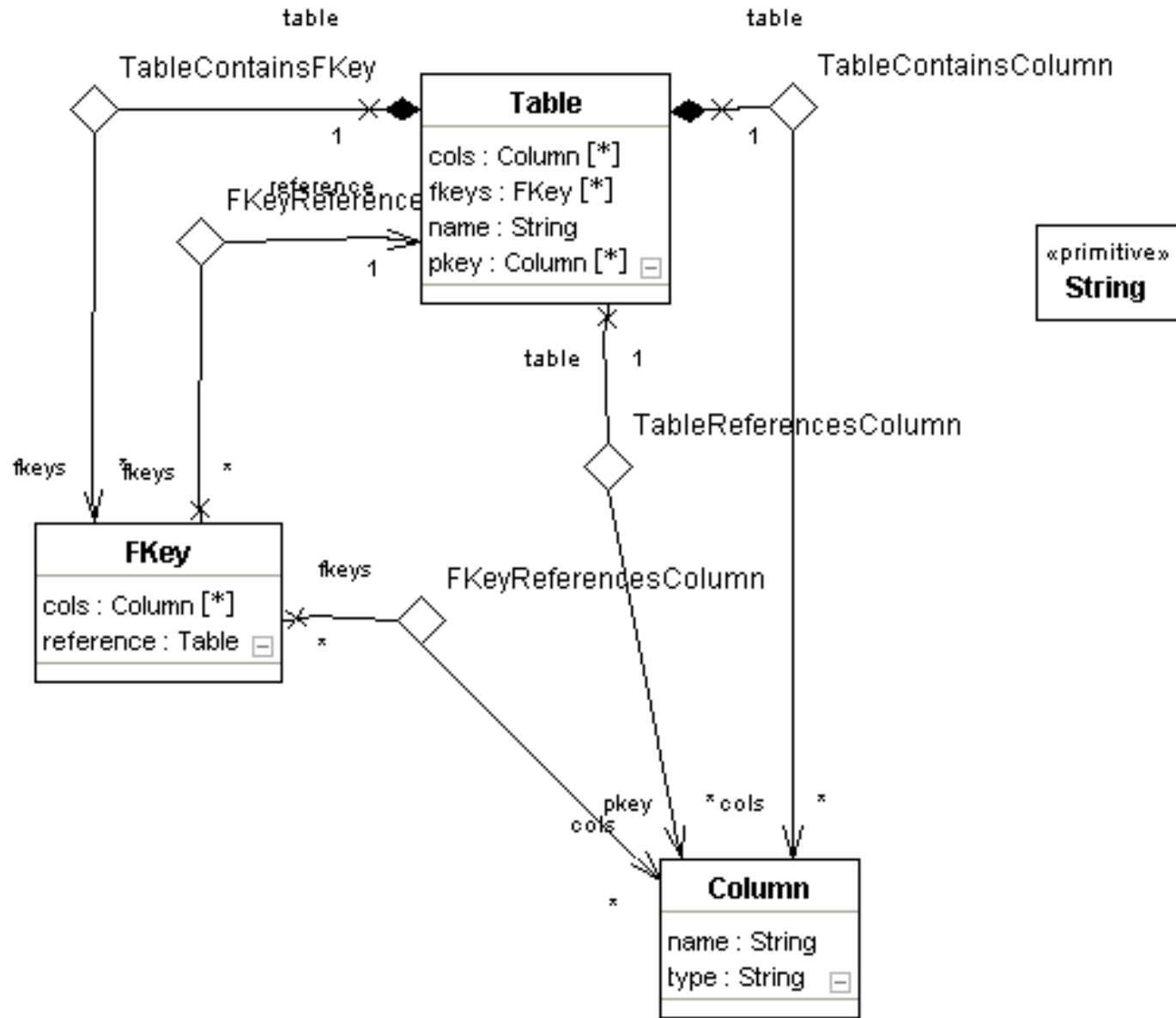
Left Metamodel: Class Diagram Metamodel (CD)

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



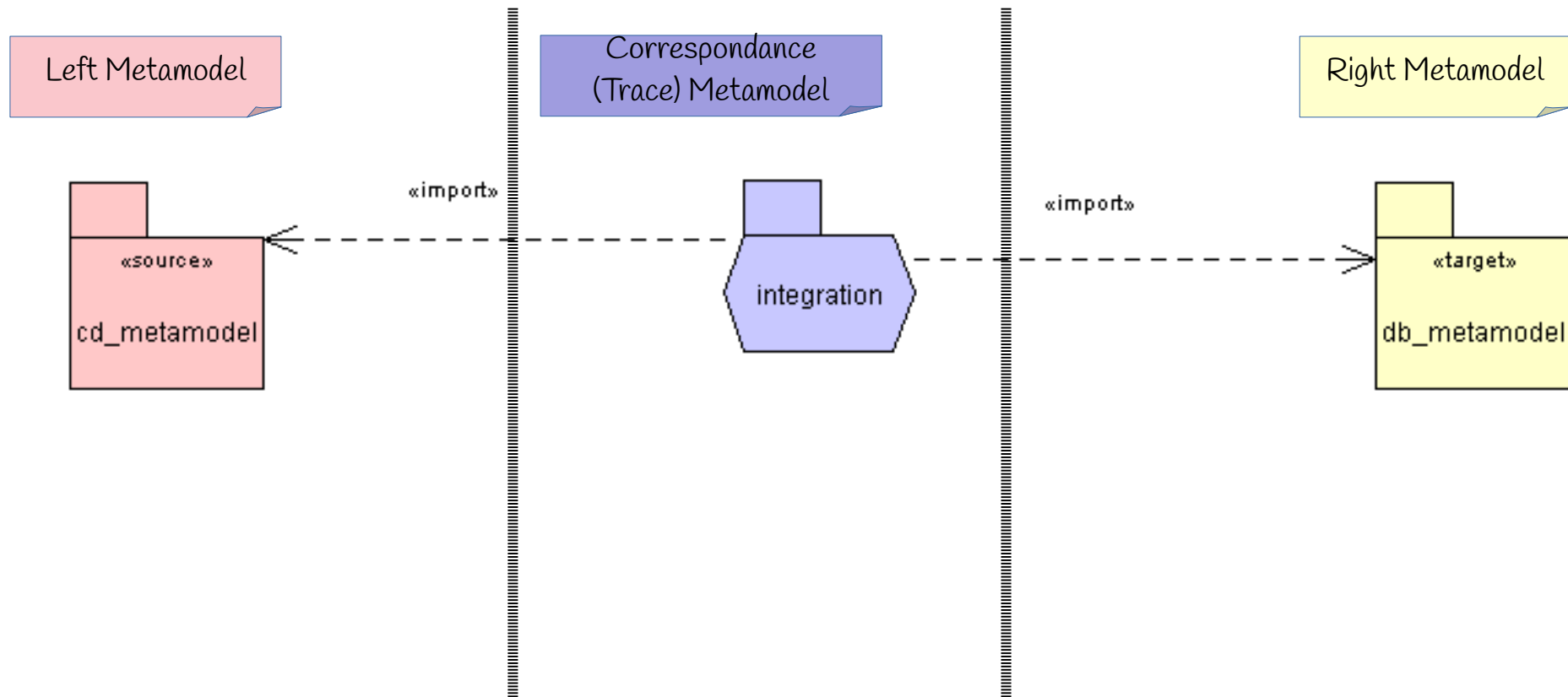
Right Metamodel: Relational Metamodel (DB, relational schema)

Right Metamodel



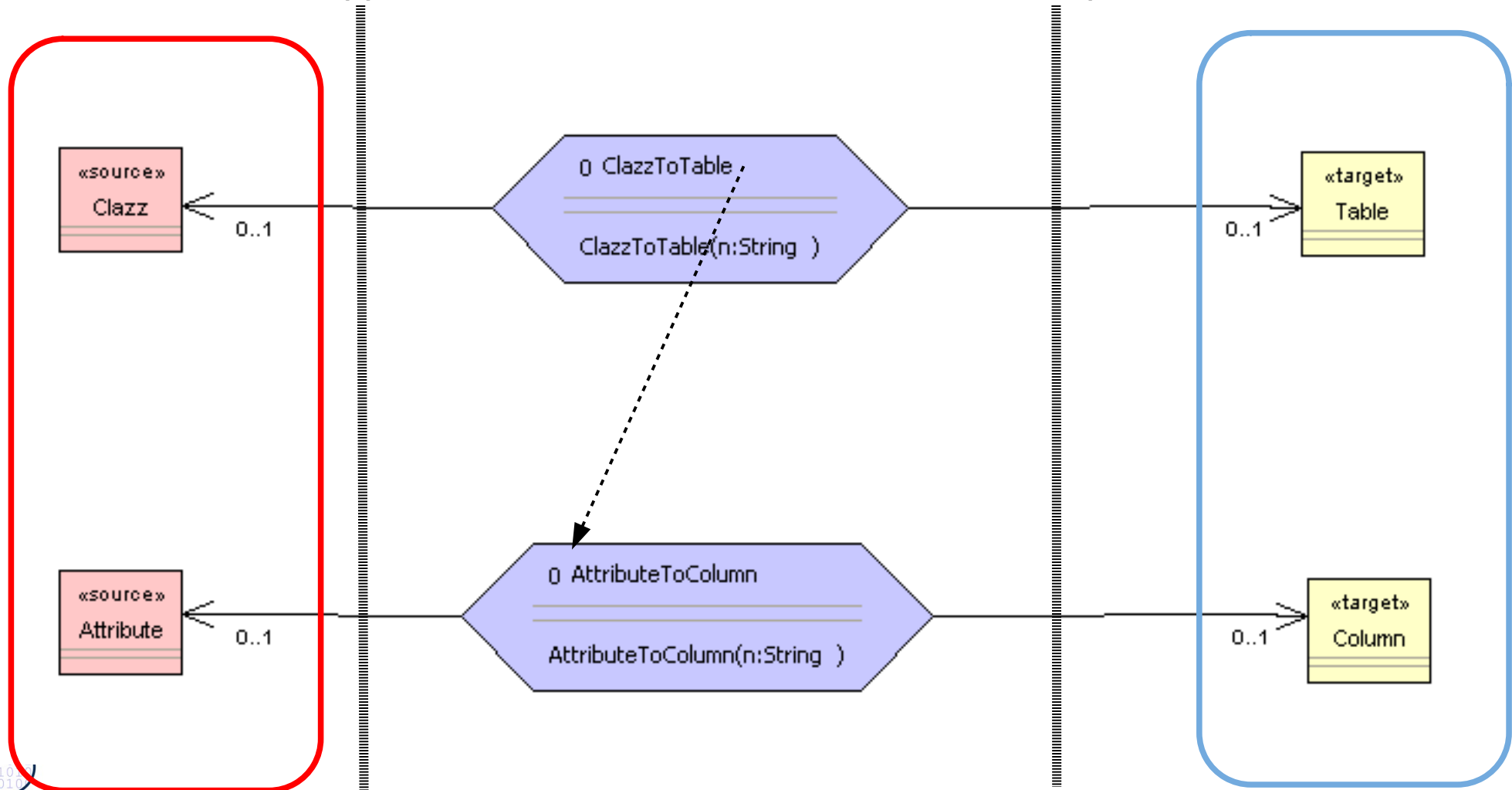
TGG for Object-Relational Mapping (ORM)

- ▶ 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.



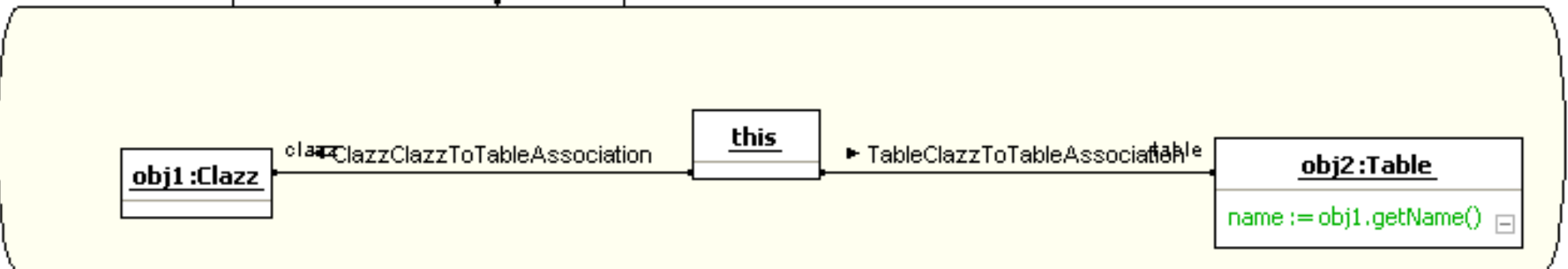
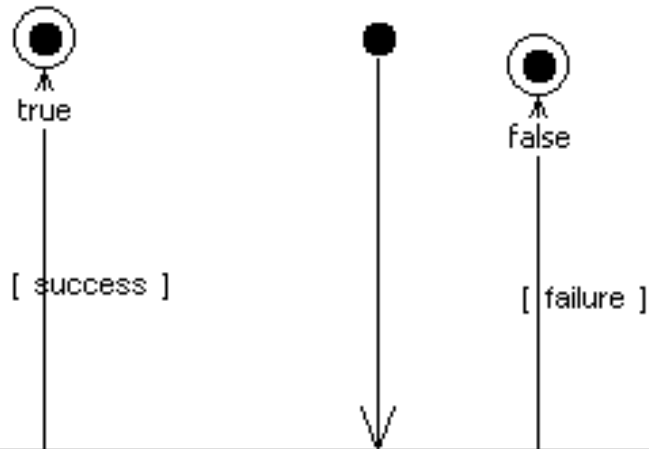
54.2. Triple Graph Grammars in MOFLON

- MOFLON in MOF Technical Space
- eMOFLON in EMOF TS

Triple Graph Grammars – Moflon Example

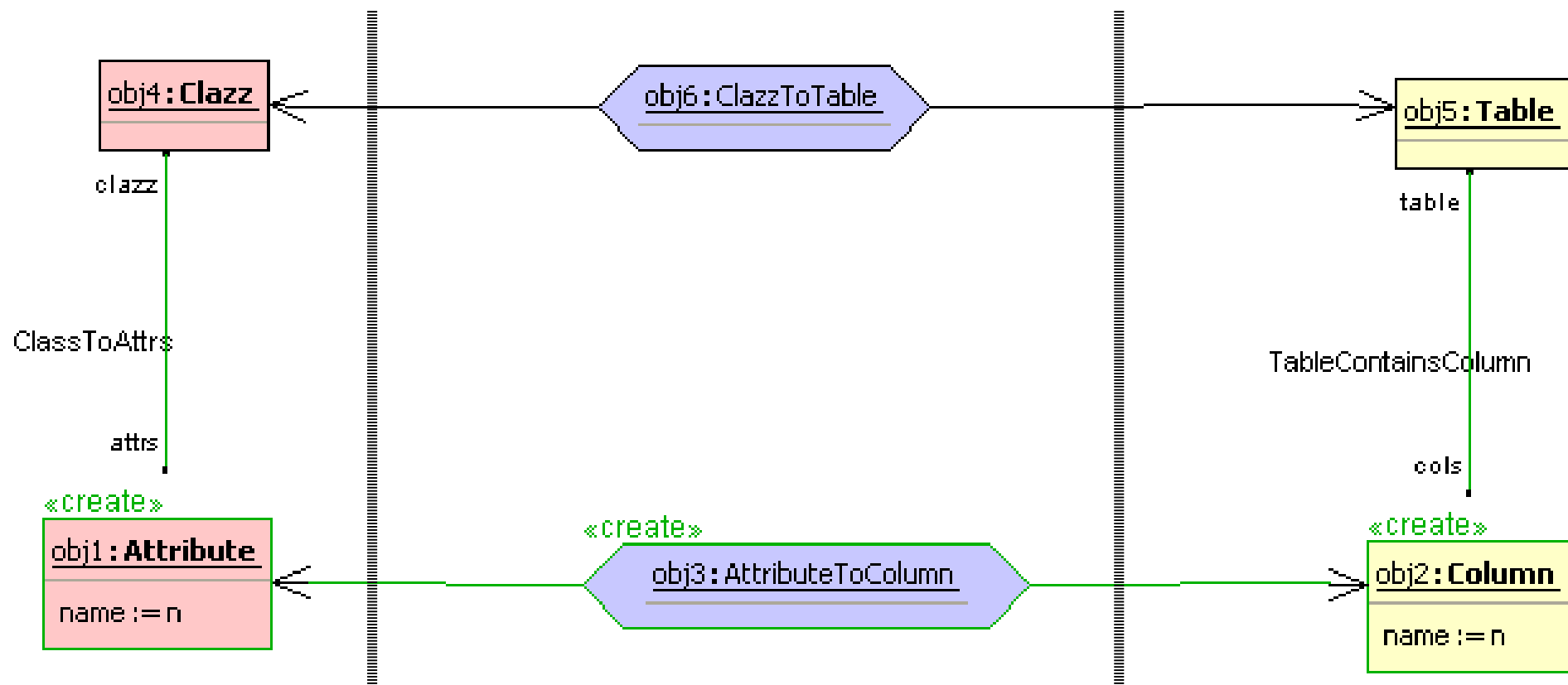
- ▶ Because they are named, TGG rules can be started by Fujaba Storyboards (activity diagrams)
- ▶ The activities can be associated to a transformation class `ClazzToTable`

`ClazzToTable::performForwardAttributeValuePropagation ()`: Boolean



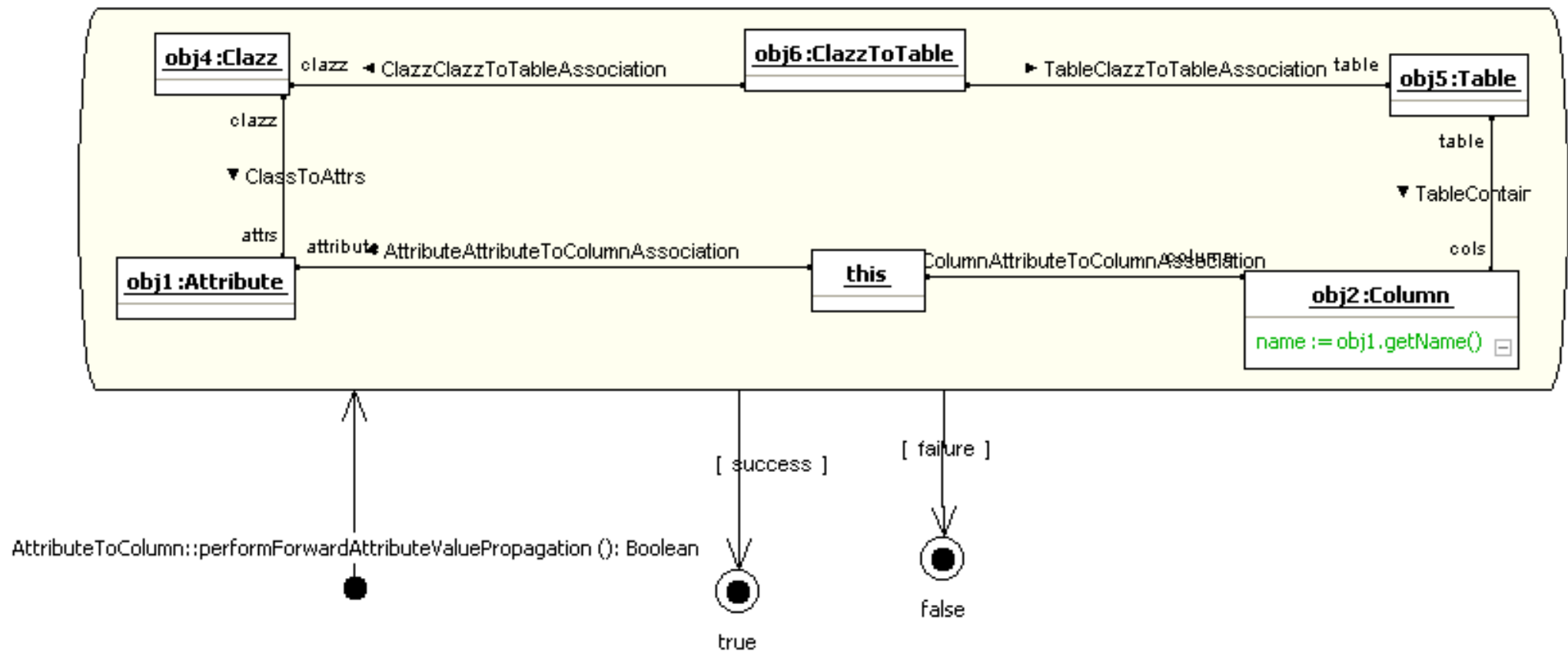
Example of Lower-Level-Creation Rule

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



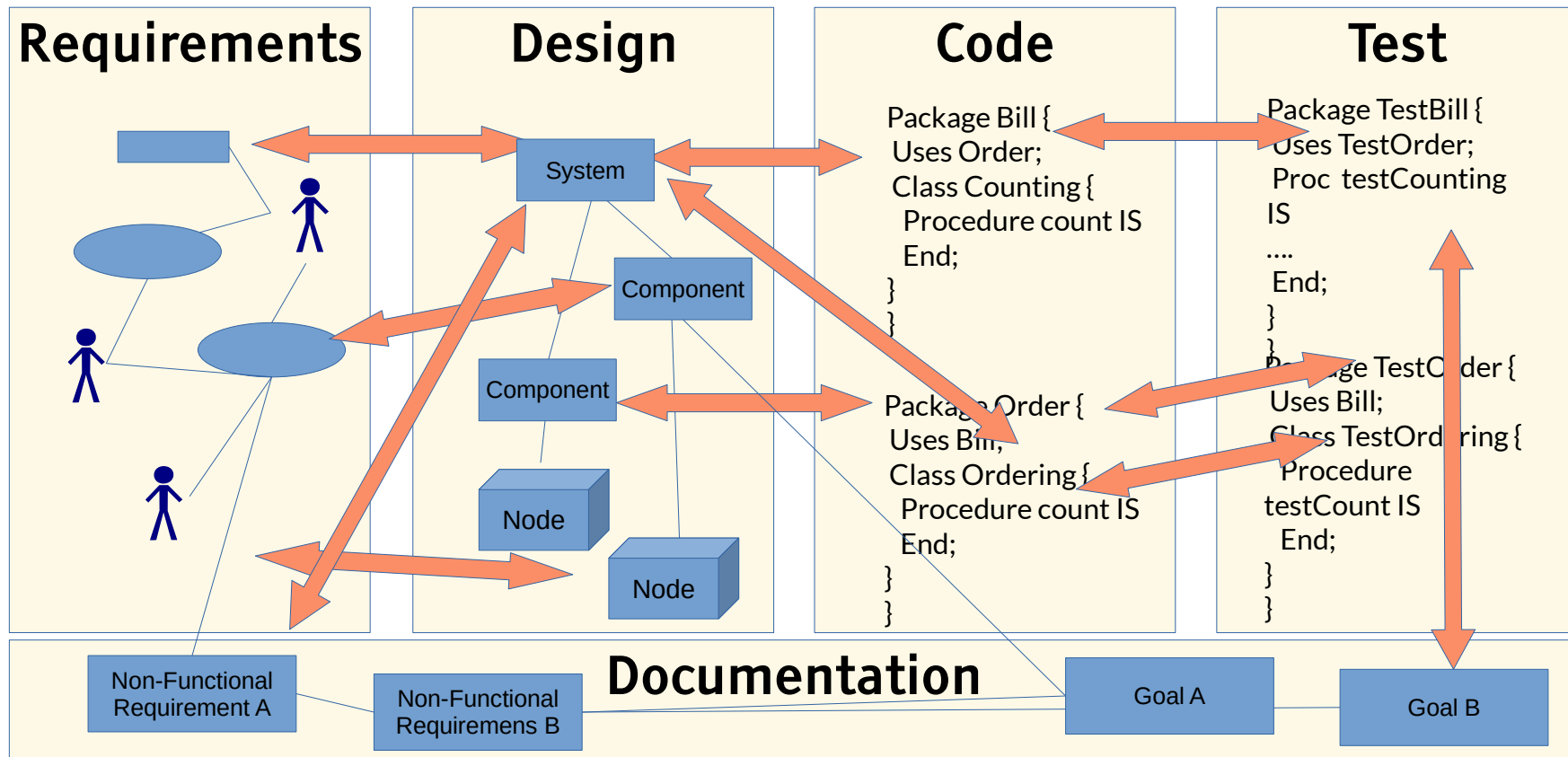
Triple Graph Grammars – Moflon Example

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

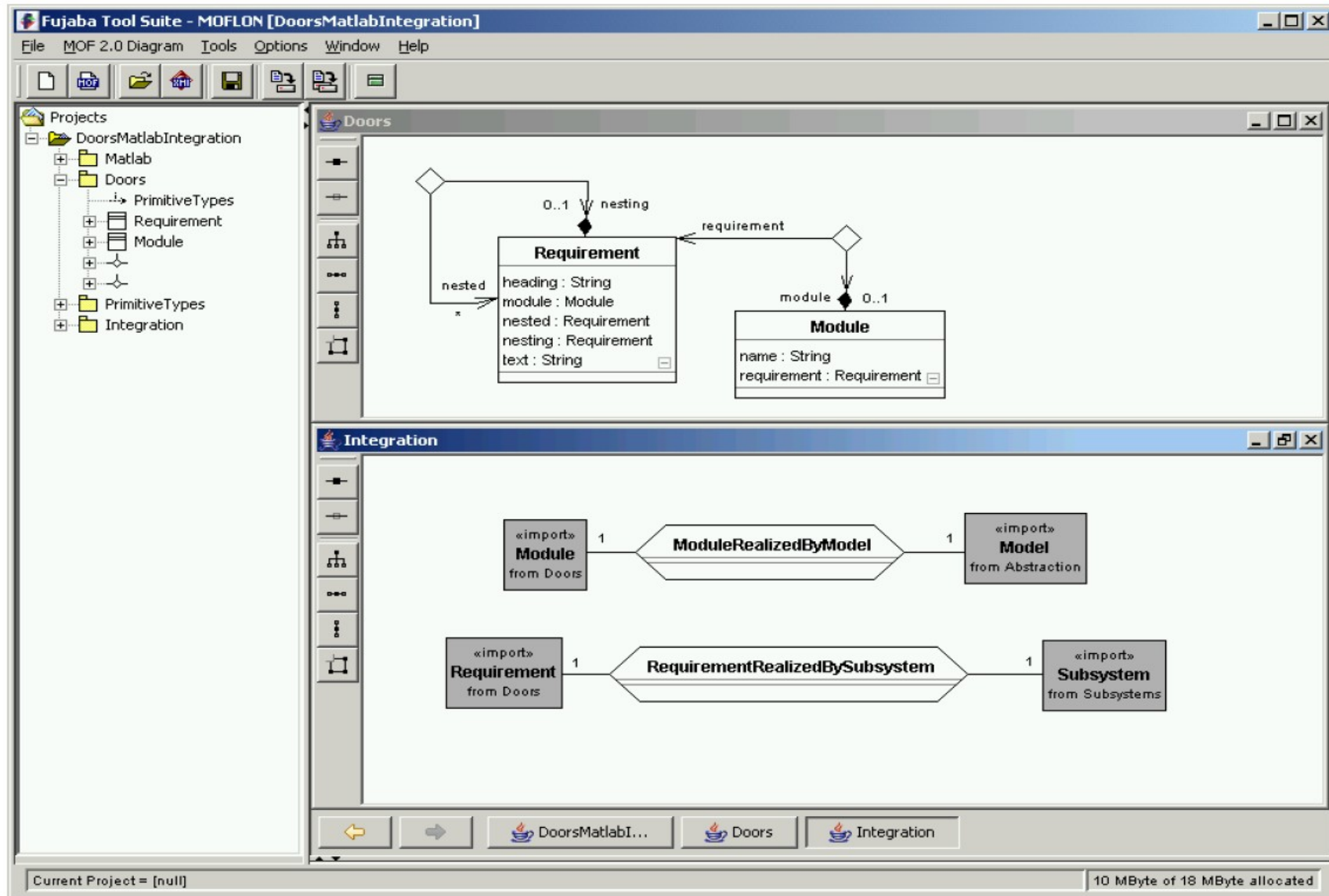


Q12: The ReDoDeCT Problem and its Macromodel

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

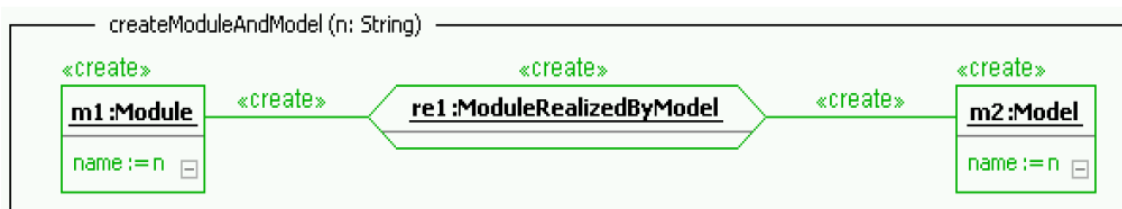


Ex. 2: TGG Coupling of Requirements Specification and Design

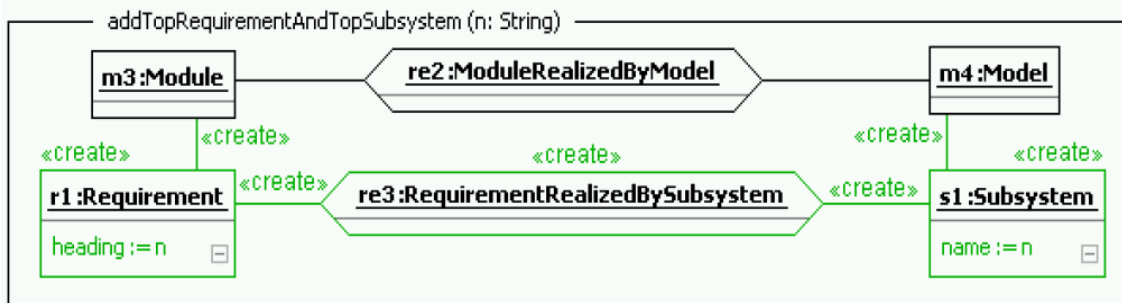


TGG Coupling Requirements Specification and Design

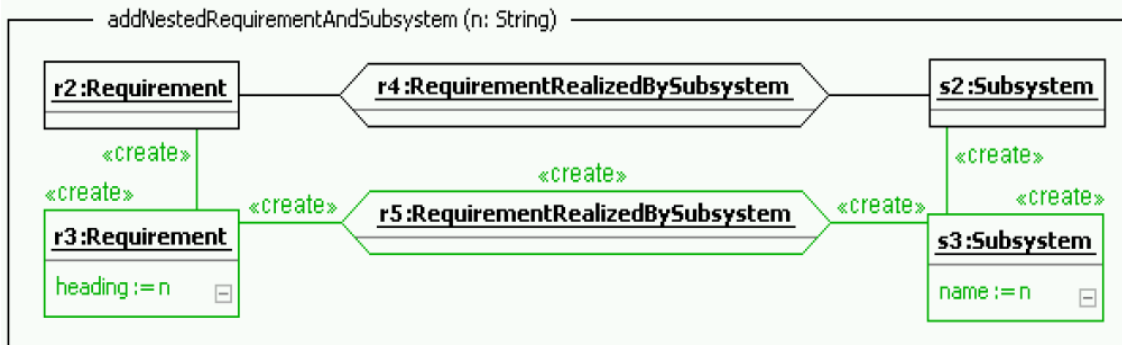
- ▶ This TGG grammar builds up a module-requirements graph
- ▶ Starting from a relation “ModuleRealizedByModel” and “RequirementRealizedBySubsystem”



initial, creational



lower-layer creational



lower-layer creational





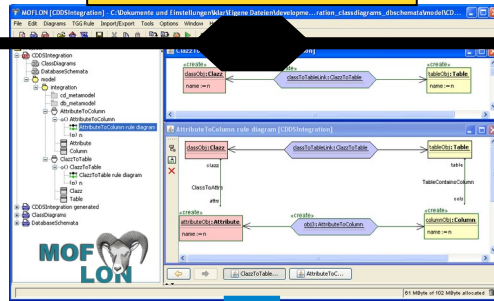
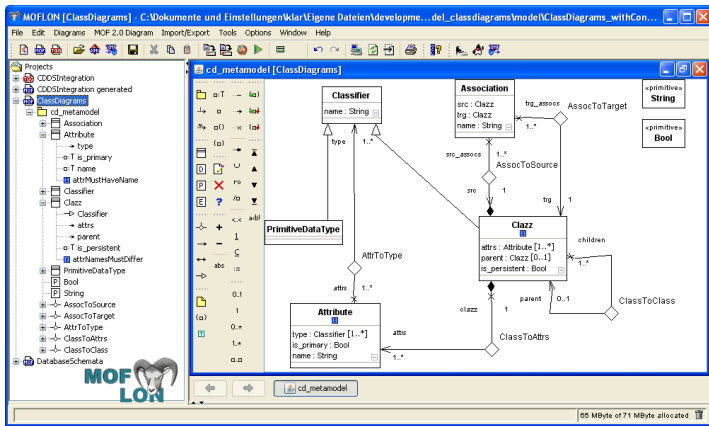
54.3. Using Triple Graph Grammars in MOFLON

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

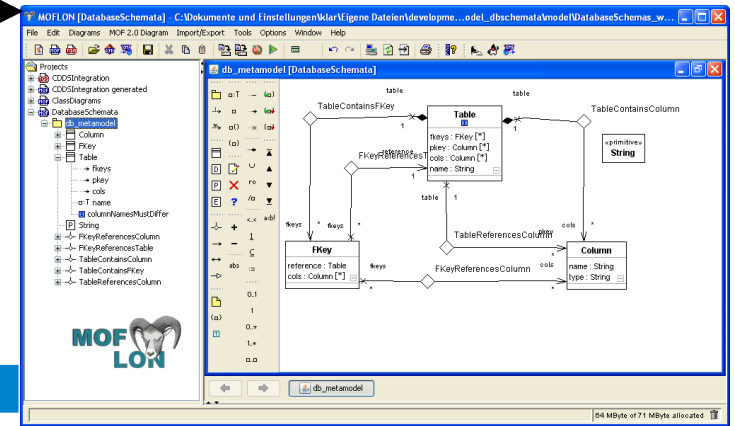
Class Diagrams Metamodel

TGGs relate

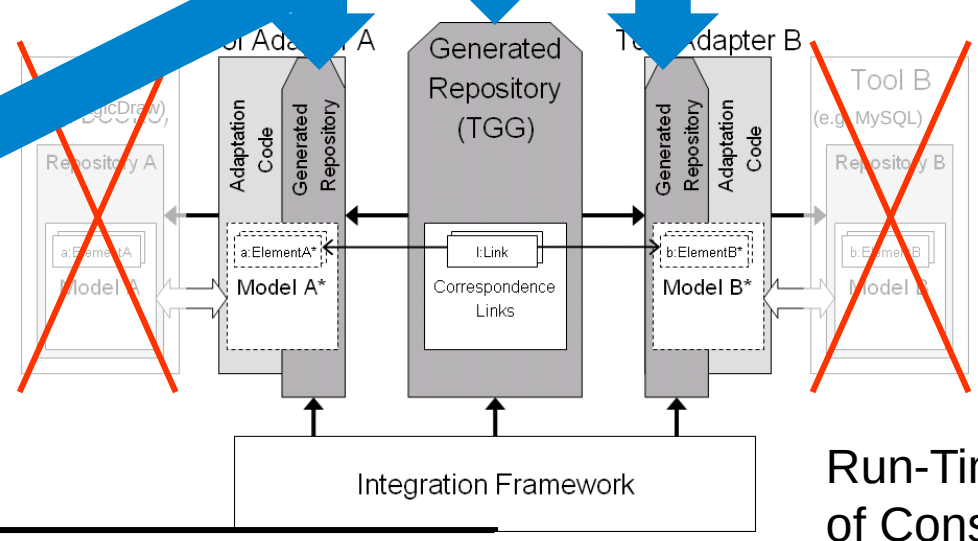
Database Schemata Metamodel



MOFLON generates



integration rule code



Run-Time Verification of Constraints



TiE-CD-DB – Constraints in Class Diagrams (1)

Generate Code from MOF model (CD metamodel)

26

Model-Driven Software Development in Technical Spaces (MOST)

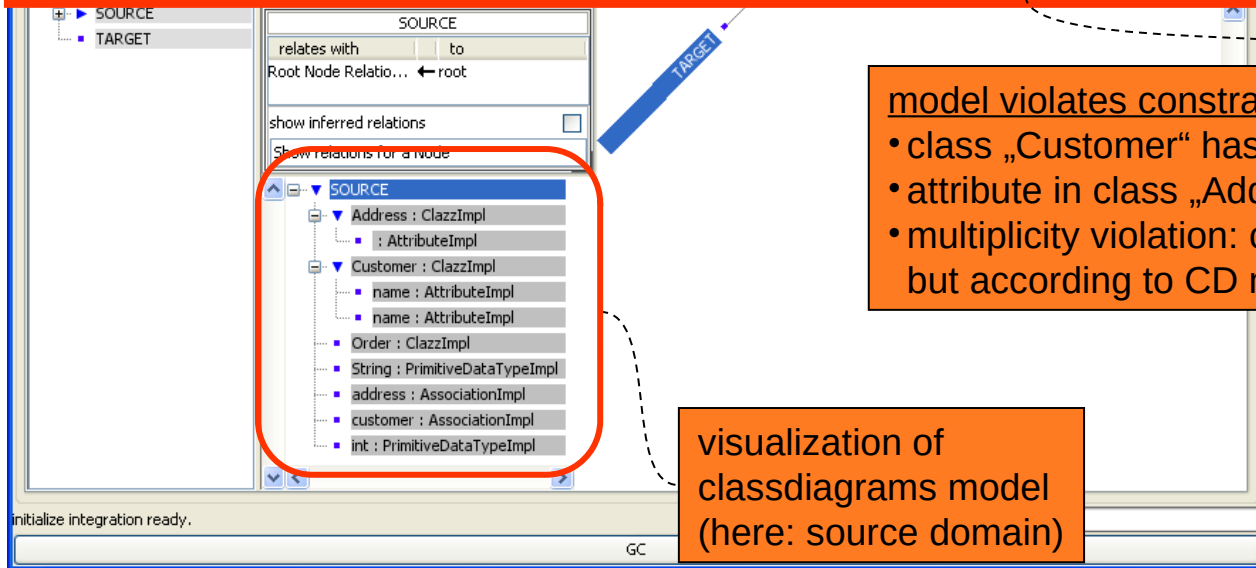
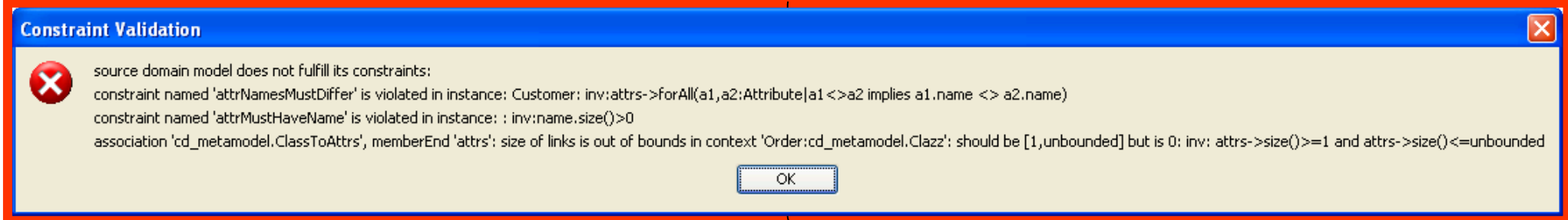
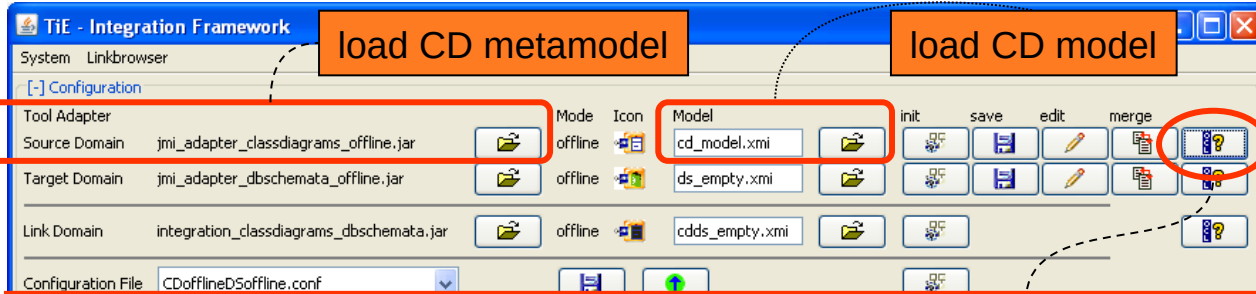
The image displays the MOFLON software interface for editing a MOF model. The main window shows a class diagram metamodel with classes: Classifier, Association, PrimitiveDataType, Attribute, andClazz. The 'Clazz' class is highlighted with a red circle. A context menu is open over the 'cd_metamodel' package, with 'Generate MOFLON-Code' selected and highlighted by a red rectangle. Below the main window, the 'Edit MOF Constraint' dialog is open, showing the constraint 'attrNamesMustDiffer' with the following OCL body:

```
inv: attrs->forAll( a1, a2 : Attribute | a1 <> a2 implies a1.name <> a2.name )
```

The dialog also shows visibility options (undefined, public, private) and language options (invariant, define).

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

Loading Metamodels and Models



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)

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

Model Browser

The screenshot displays the TiE - Integration Framework interface on the left and the JmiModelBrowser on the right. The JmiModelBrowser shows a hierarchical tree of model elements under 'cd_metamodel', including classes like 'customer:AssociationImpl', 'Order:ClazzImpl', 'Customer:ClazzImpl', 'Address:ClazzImpl', and primitive data types like 'int:PrimitiveDataTypeImpl' and 'String:PrimitiveDataTypeImpl'. The 'surname:AttributeImpl' element is selected. A 'String Editor Dialog' is open, showing a text input field with 'surname' and 'OK'/'Abbrechen' buttons. Below the dialog, a table shows constraints for the 'name' attribute.

name	value	edit
name	surname	edit
is_primary	false	edit
type	set[String]	edit

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

model is fixed in generic model editor

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

System Linkbrowser

[-] Configuration

Tool Adapter	Mode	Icon	Model	init	save	edit	merge
Source Domain: jmi_adapter_classdiagrams_offline.jar	offline		cd_model.xmi				
Target Domain: jmi_adapter_db schemata_offline.jar	unknown		ds_empty.xmi				
Link Domain: integration_classdiagrams_db schemata.jar	unknown		cdds_empty.xmi				

Configuration File: CDofflineDSoffline.conf

[-] Action

Algorithm: Forward Translation (Batch, Simple) Strategy: Unsorted Simple Log Level: WARN

Configuration File: last.conf

[-] Output

LinkBrowser Log

root

- SOURCE
- TARGET

Close Up View CircleView

relates with to

show inferred relations

Show relations for a Node

SOURCE

- Address : ClassImpl
 - street : AttributeImpl
- Customer : ClassImpl
 - name : AttributeImpl
 - surname : AttributeImpl
- Order : ClassImpl
 - id : AttributeImpl
- String : PrimitiveDataTypeImpl
- address : AssociationImpl
- customer : AssociationImpl

initialize source ready.

GC

translation process may start now...

Constraint Validation

source domain model fulfills its constraints

OK

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

Forward Translation to DB representation

The image displays two screenshots of the TiE - Integration Framework software, illustrating the forward translation process from class diagrams to a database representation.

Left Screenshot: Configuration and Action Settings

- System Linkbrowser:** Shows the configuration for the tool adapter, source domain, target domain, and link domain.
- Configuration File:** Set to `CDofflineDSoffline.conf`.
- Action:** Algorithm is `Forward Translation (Batch, Simple)` and Strategy is `Unsorted Simple`.
- Configuration File:** Set to `last.conf`.
- Output:** Shows a LinkBrowser with a tree view containing `SOURCE` and `TARGET` nodes.

Right Screenshot: Output of the Translation

- System Linkbrowser:** Shows the configuration for the tool adapter, source domain, target domain, and link domain.
- Configuration File:** Set to `CDofflineDSoffline.conf`.
- Action:** Algorithm is `Forward Translation (Batch, Simple)` and Strategy is `Unsorted Simple`.
- Configuration File:** Set to `last.conf`.
- Output:** Shows a LinkBrowser with a tree view containing `SOURCE` and `TARGET` nodes. The `TARGET` node is expanded, showing a class diagram with the following elements:
 - `Address : ClassImpl` (with attribute `street : AttributeImpl`)
 - `Customer : ClassImpl` (with attributes `name : AttributeImpl` and `surname : AttributeImpl`)
 - `Order : ClassImpl` (with attribute `id : AttributeImpl`)
 - `String : PrimitiveDataTypeImpl` (with attribute `address : AssociationImpl`)
 - `int : PrimitiveDataTypeImpl` (with attribute `customer : AssociationImpl`)

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

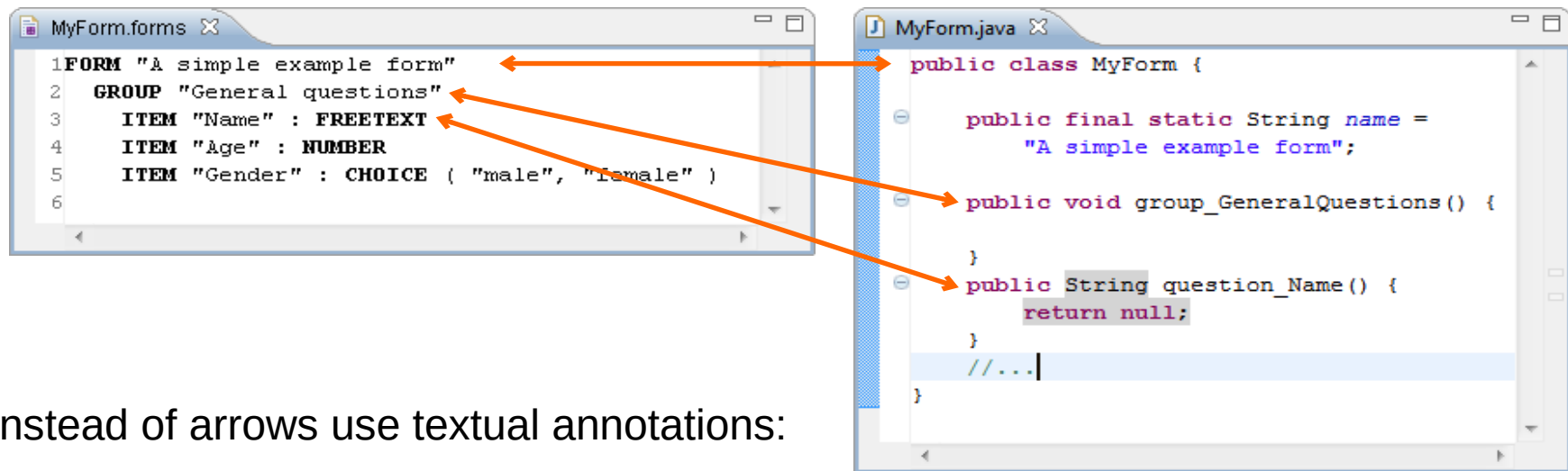
Why?

- ▶ 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 EMFTText

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



The image shows two side-by-side code editors. The left editor, titled 'MyForm.forms', contains the following text:

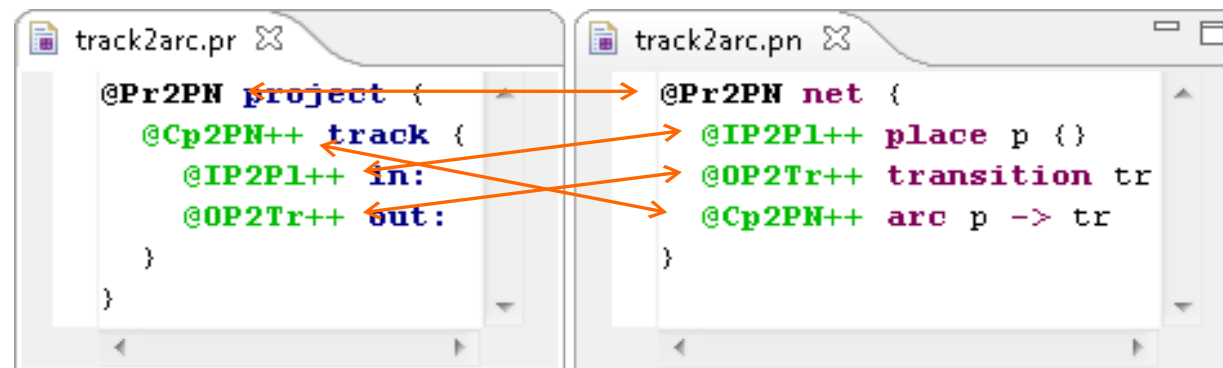
```
1FORM "A simple example form"
2  GROUP "General questions"
3    ITEM "Name" : FREETEXT
4    ITEM "Age" : NUMBER
5    ITEM "Gender" : CHOICE ( "male", "female" )
6
```

The right editor, titled 'MyForm.java', contains the following Java code:

```
public class MyForm {
    public final static String name =
        "A simple example form";
    public void group_GeneralQuestions() {
    }
    public String question_Name() {
        return null;
    }
    //...
}
```

Four orange arrows point from the forms editor to the Java editor: one from line 1 to the class declaration, one from line 2 to the group method, one from line 3 to the question_Name method, and one from line 5 to the return null statement.

Instead of arrows use textual annotations:



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

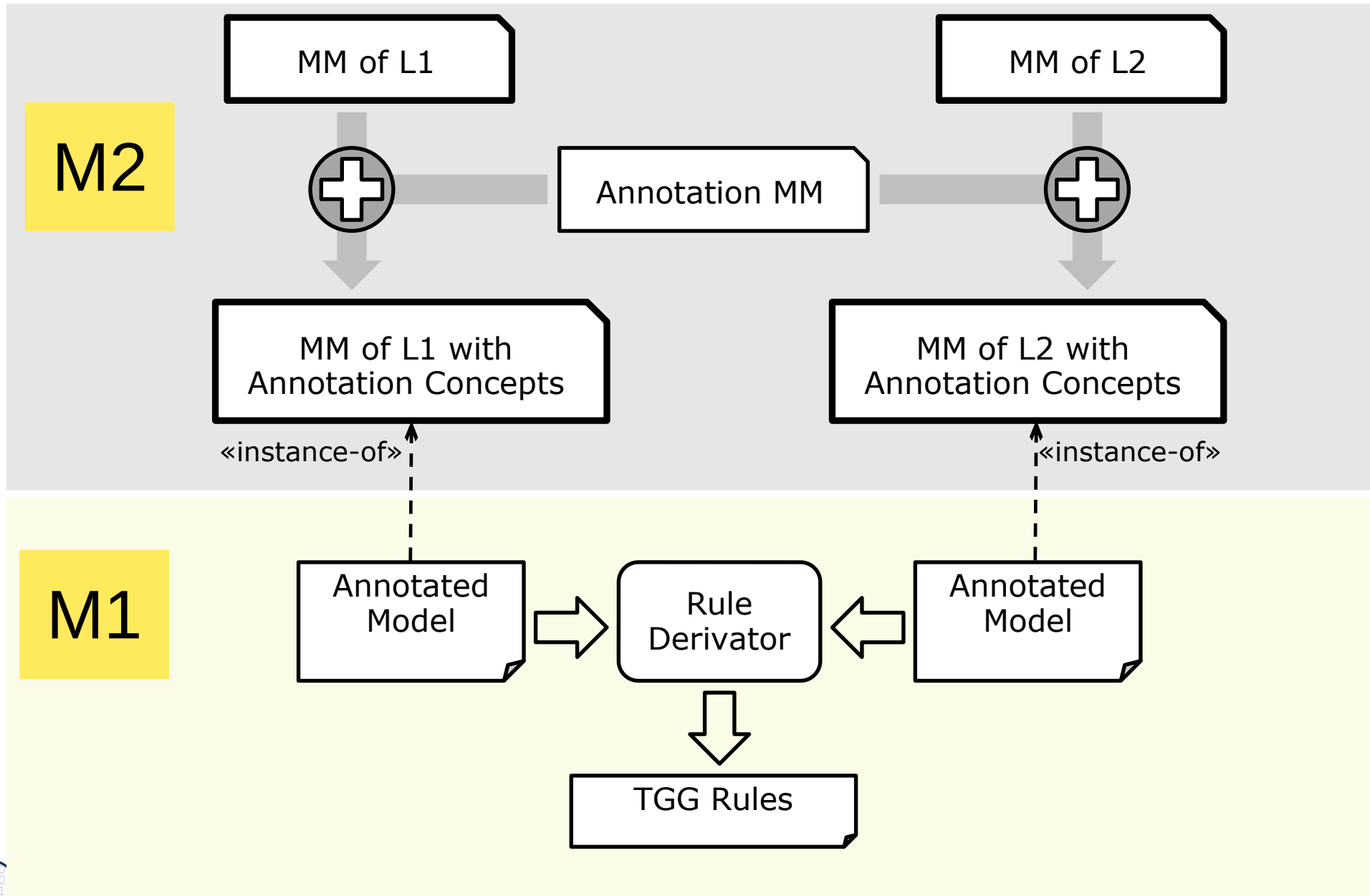
```
@Pr2PN project {
  @Cp2PN++ track {
    @IP2P1++ in:
    @OP2Tr++ out:
  }
}
```

The right editor, titled 'track2arc.pn', contains the following Petri net code:

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

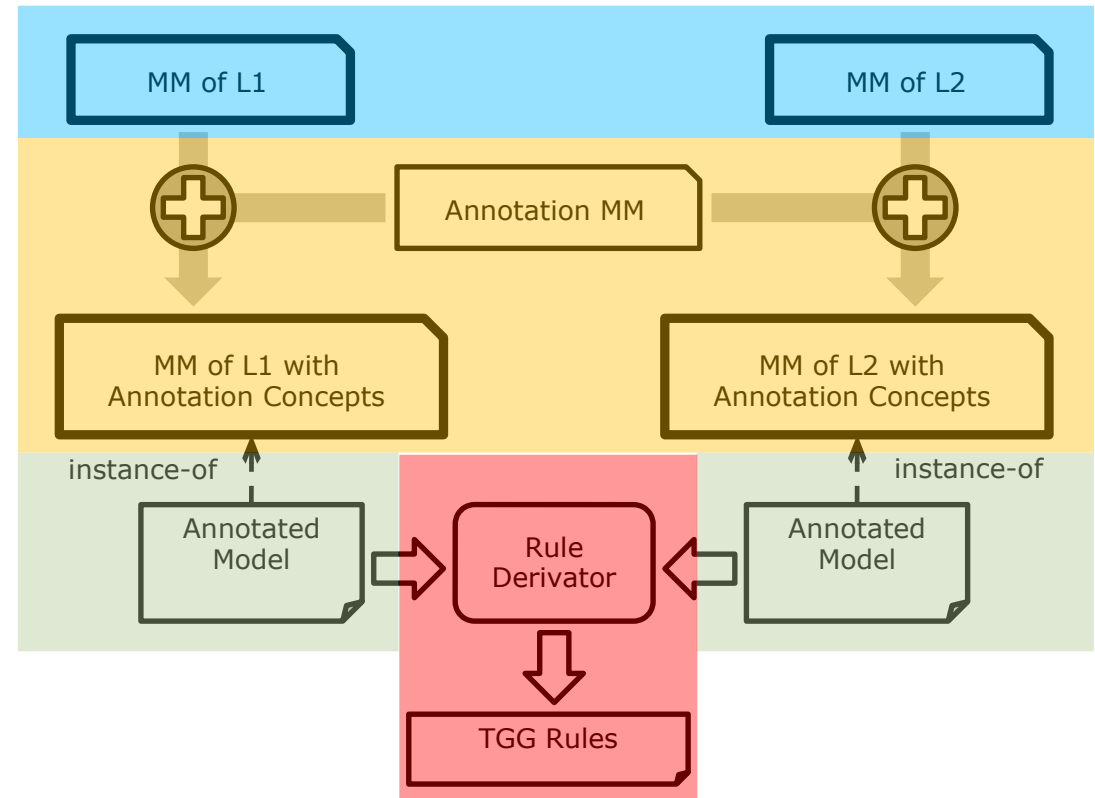
Four orange arrows point from the Petri net code to the project code: one from '@Pr2PN net' to '@Pr2PN project', one from '@IP2P1++ place p' to '@IP2P1++ in:', one from '@OP2Tr++ transition tr' to '@OP2Tr++ out:', and one from '@Cp2PN++ arc p -> tr' to '@Cp2PN++ track'.

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



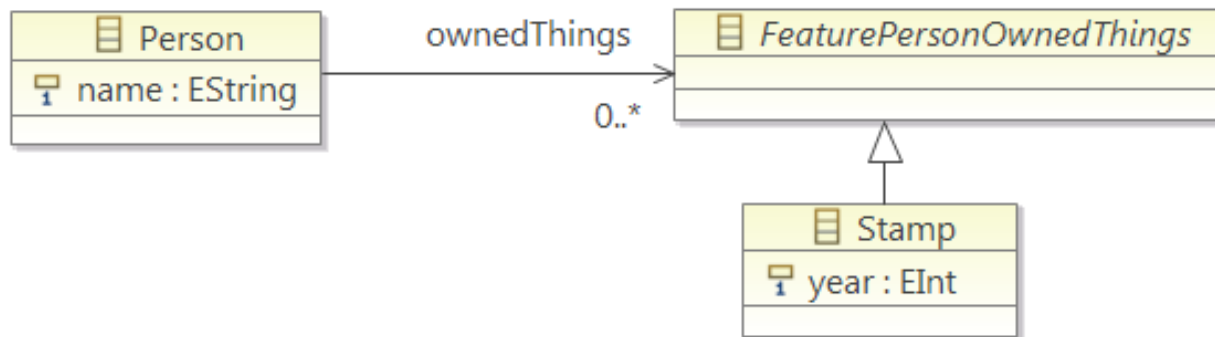
Step 1a – Getting Extensible Metamodels

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:



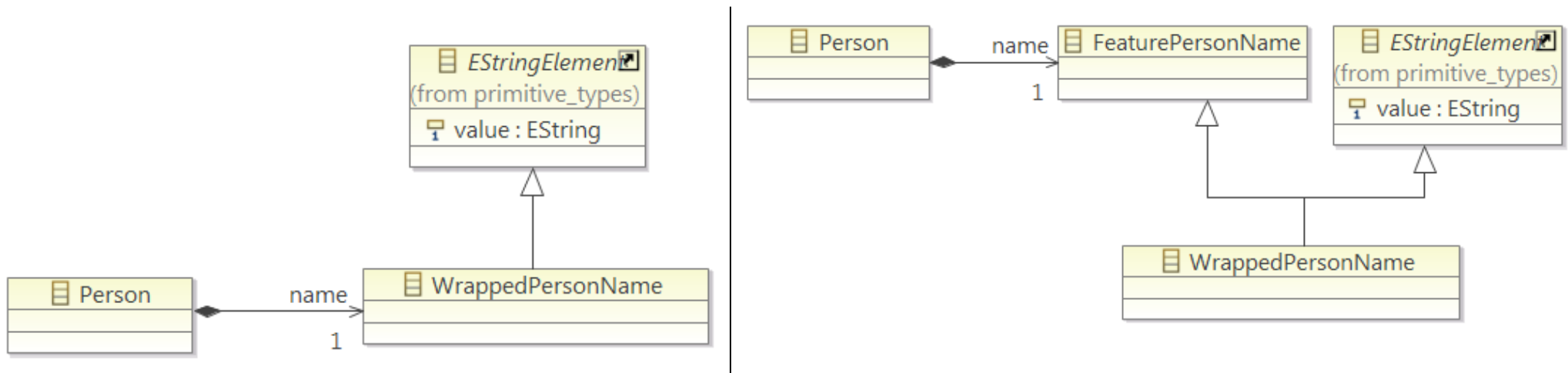
Step 1b – Getting Extensible Metamodels

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:



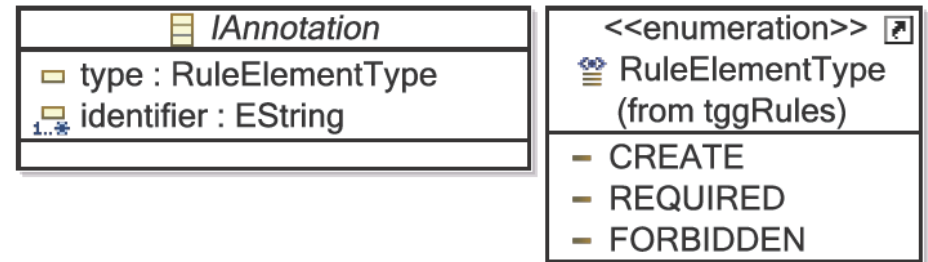
Wrapped attribute

Wrapped attribute after step 1a)

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)

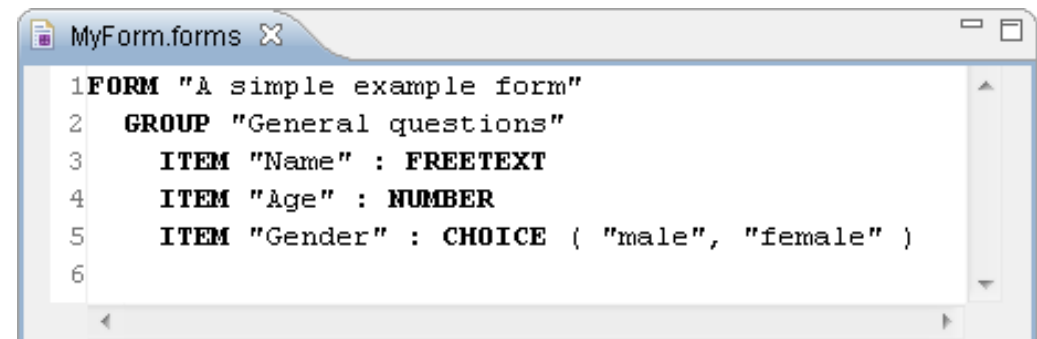
Step 3 – Extending the Concrete Syntax Specification

- ▶ 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;
```

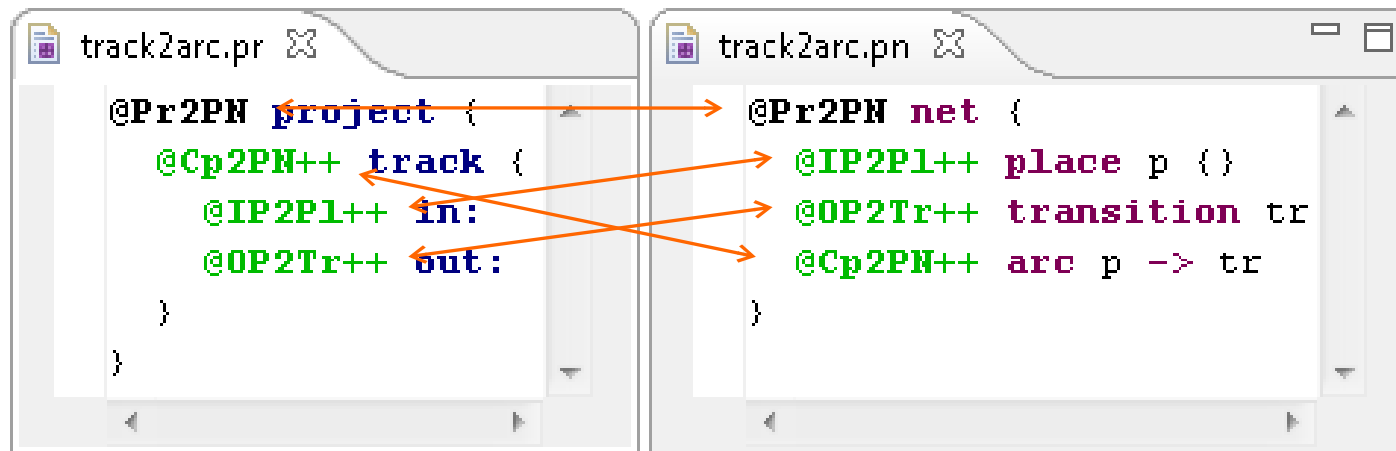
`@1 !` `FORM "A simple example form"`

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



Step 3 – Extended Concrete Syntax

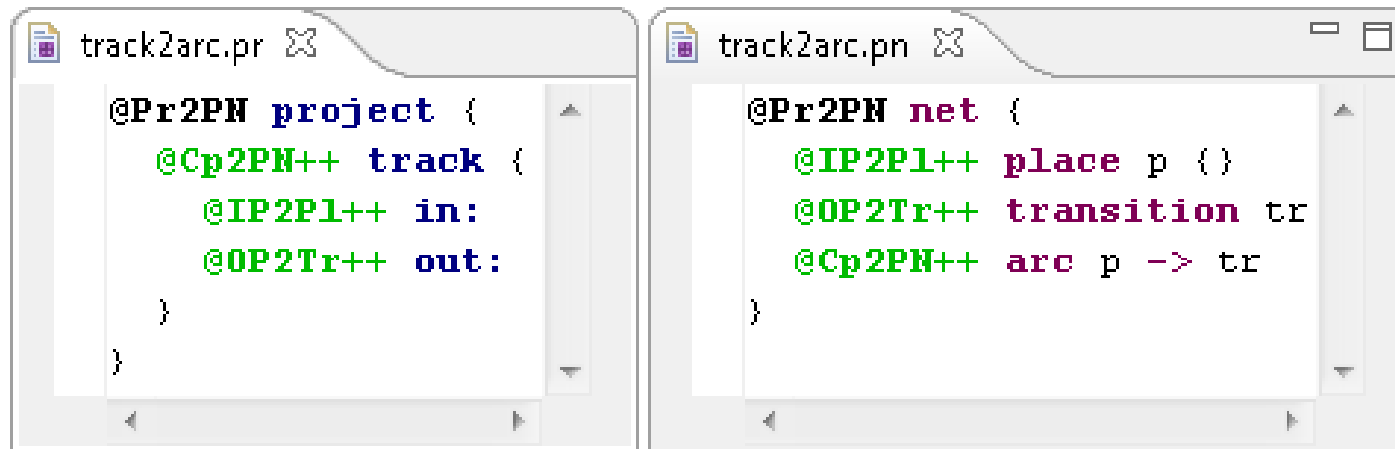
Rail tracks to Petrinet example



(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 editors. The left editor, titled 'track2arc.pr', contains the following code:

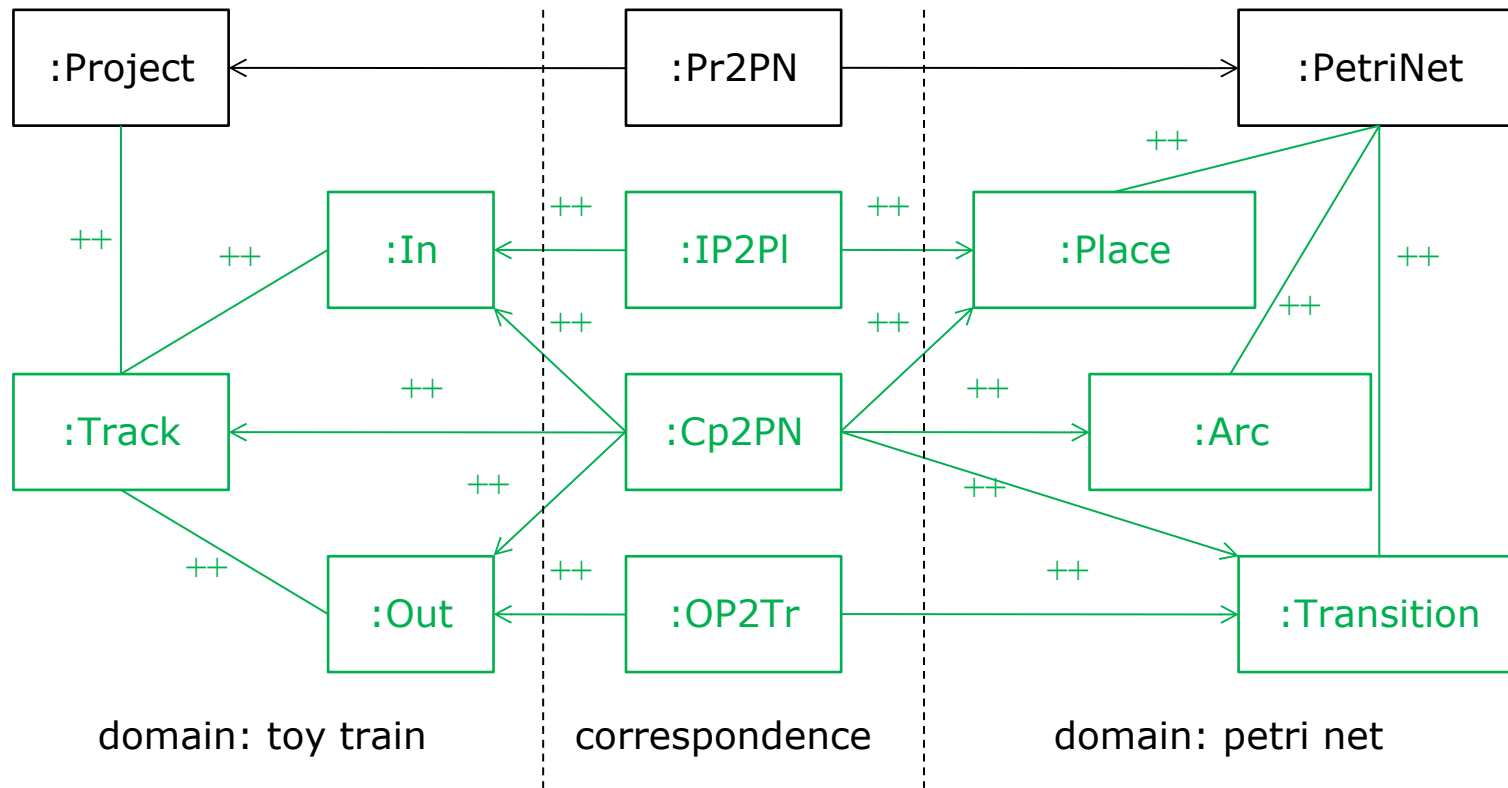
```
@Pr2PN project {  
  @Cp2PN++ track {  
    @IP2P1++ in:  
    @OP2Tr++ out:  
  }  
}
```

The right editor, 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 Petrinet 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