

Model Transformation

Krzysztof Czarnecki
Generative Software Development Lab
University of Waterloo, Canada
gsd.uwaterloo.ca

Modeling Wizards Summer School, Oct. 1, 2010, Oslo, Norway

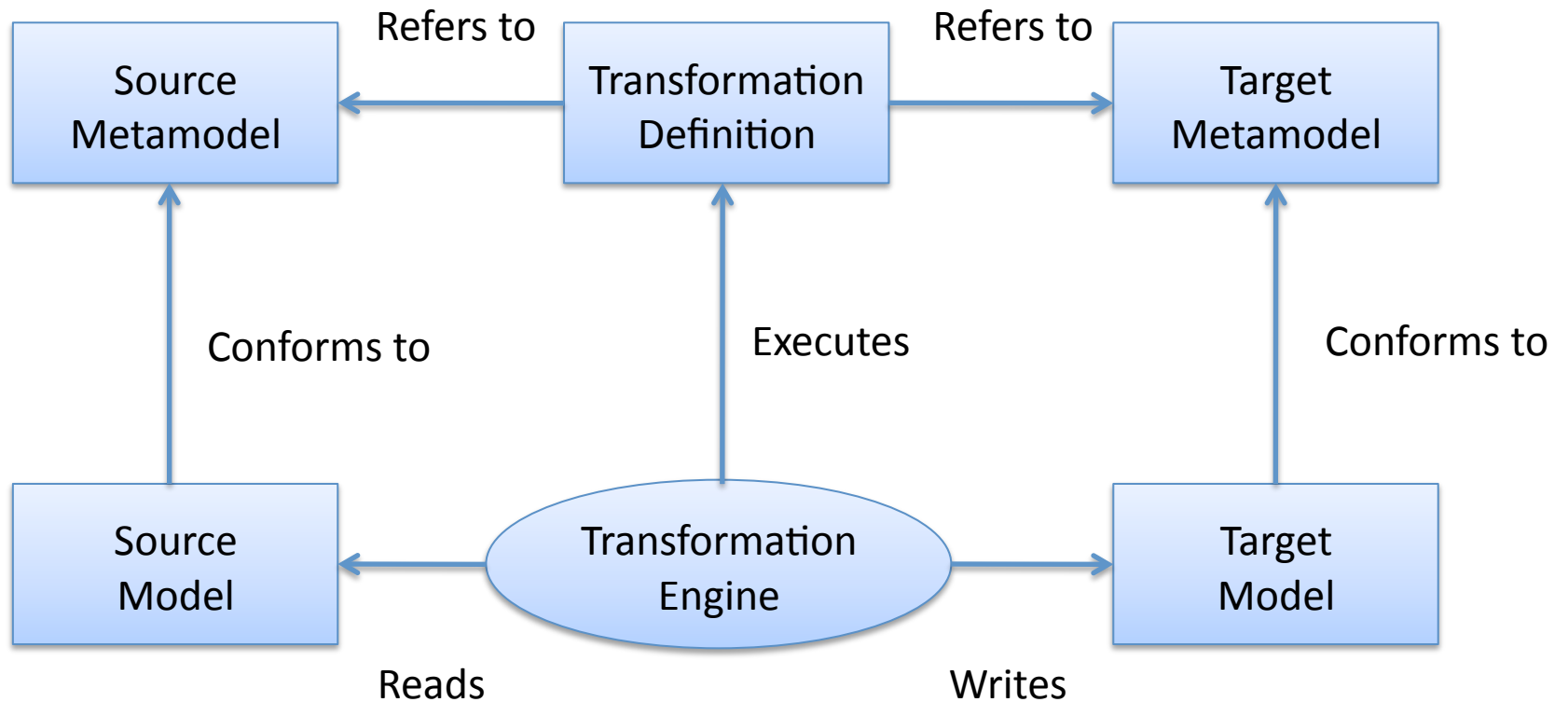
What is model transformation?

- Model manipulation
 - Automated reading, creation, modification
 - Form of metaprogramming
- Program vs. model transformation
 - Blurred boundary
 - Program vs. model
 - String, trees, graphs
 - Grammars vs. class models
 - Model trafo is more diverse

Applications

- Deriving lower-level models from higher-level ones
 - Compilation – automatic
 - Generating designs – partially automated
- Mapping and synchronizing among models at same or across abstraction levels
- Creating query-based views of a system
- Model evolution, e.g., refactoring
- Reverse engineering higher-level models from lower-level ones
- Transforming between different formats or languages
 - E.g., tool integration

Basic concepts



Development of Transformations

- Basic ingredients
 - Metamodel and internal representation
 - Queries
 - Target element creation
 - Logic (execution: traversal, scheduling, ...)
- Mapping (Specification) vs Transformation (Implementation)
- Verification and validation

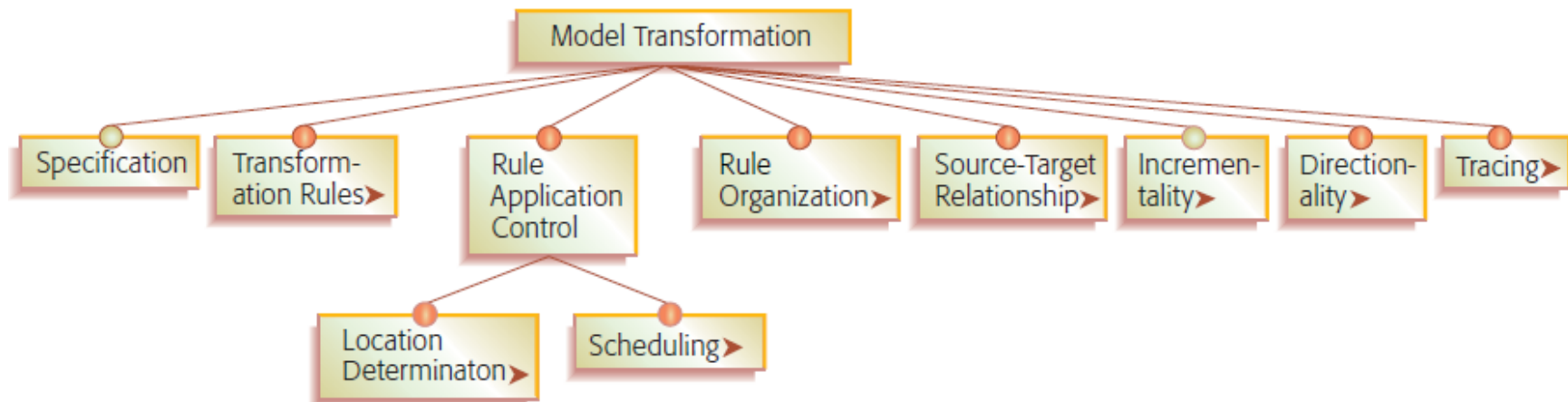
Best Practice

- Augment the source metamodel with a library of queries
- Reuse such a library if available

Transformation Systems and Languages

- Huge number of systems and growing...
 - Over 30 systems and languages in the 2006 survey
 - Vast majority academic
 - A few industrial-strength (improved somewhat compared to 2006)
- Very diverse in terms of paradigms, features, and capabilities
 - Choice depends on application

Classification of language features



Czarnecki, K. & Helsen, S.

Feature-Based Survey of Model Transformation Approaches.

IBM Systems Journal, special issue on Model-Driven Software Development,
45, no. 3, pp. 621-645, July 2006

Download at <http://gsd.uwaterloo.ca/node/68>

Language Features I

- Query
 - Explicit navigation
 - Source patterns
 - Anstract vs. concrete syntax
 - Graphs, trees, strings
- Element creation
 - Explicit creation
 - Target patterns
- Rules combine source and target patterns

Language Features II

- Logic
 - Language paradigm
 - OO, functional, logic, procedural
- Arity and mode
 - Number of participating models (1, 2, 3, ...)
 - In, out, inout
- Directionality
 - One-way, bi-directional
- Source-target relation
 - New target (extract) vs. in-place (destructive or extension only)

Language Features III

- Traceability
 - Built-in or custom; creation, storage
- Incrementality
 - Target, source
- User-edit preservation
 - Protected regions, separation, update policy
- Organization and reuse
 - Source, target, independent
 - Modules, calls, inheritance, polymorphism, aspects

Major Categories

- Model-to-text
 - Traverse and print
 - Example: roll your own visitor pattern in Java
 - Template-based (“text with holes”)
 - E.g., Xpand, JET, (other: MOF2Text, MOFScript)
- Model-to-model
 - ...
- Text-to-model
 - Parsing
 - E.g., XText

Xpand Template

```
«DEFINE Root FOR data::DataModel»  
    «EXPAND Entity FOREACH entity»  
«ENDDEFINE»
```

```
«DEFINE Entity FOR data::Entity»  
    «FILE name + ".java"»  
    public class «name» {  
        «FOREACH attribute AS a»  
            private «a.type» «a.name»;  
        «ENDFOREACH»  
    }  
    «ENDFILE»  
«ENDDEFINE»
```

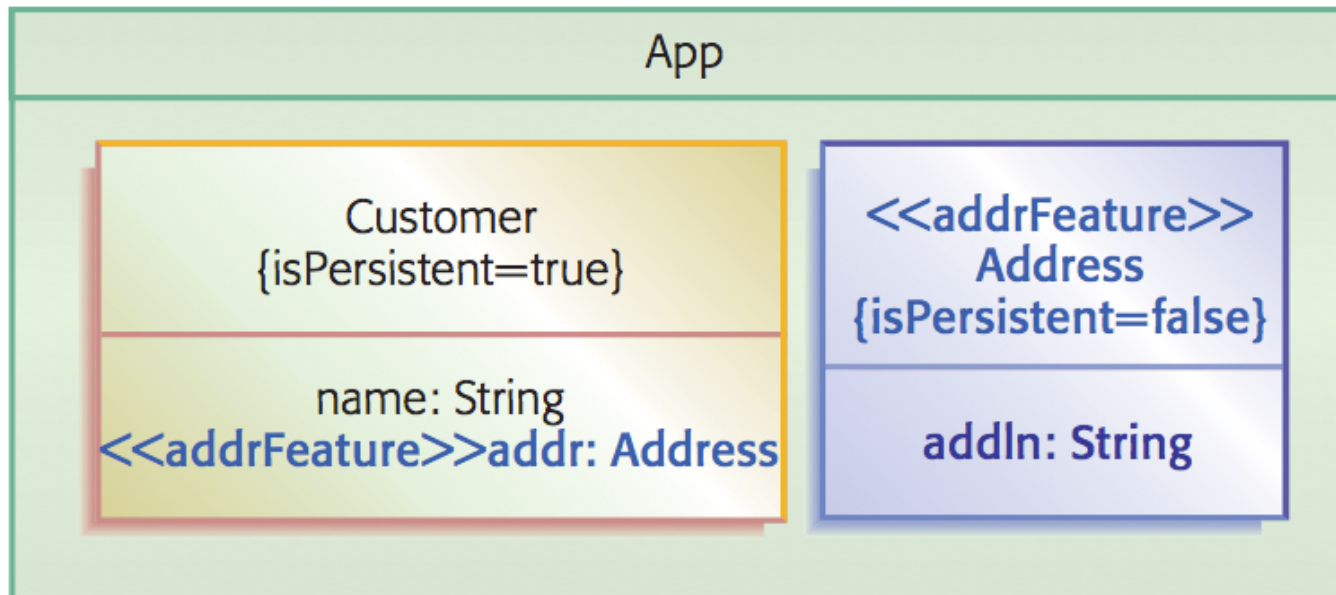
Model-to-model (I)

- Direct manipulation
 - Just use a programming language, e.g., Java
 - Often an API over the metamodels
- Operational
 - Programming language with dedicated facilities, e.g., query, pattern matching
 - Often an extension of the metamodeling formalism
 - Examples: Kermeta
- Template-based
 - Models with holes
 - Optional parts and fragment replacement
 - Examples: Model Templates [CA06], CVL

Kermeta (operational)

```
operation transform(source:PackageHierarchy): DataBase is do
  result := DataBase.new
  trace.initStep("uml2db")
  source.hierarchy.each{ pkg |
    var scm: Schema init Schema.new
    scm.name := String.clone(pkg.name)
    result.schema.add(scm)
    trace.addlink("uml2db", "package2schema", pkg, scm)
  }
end
```

Sample Model Template



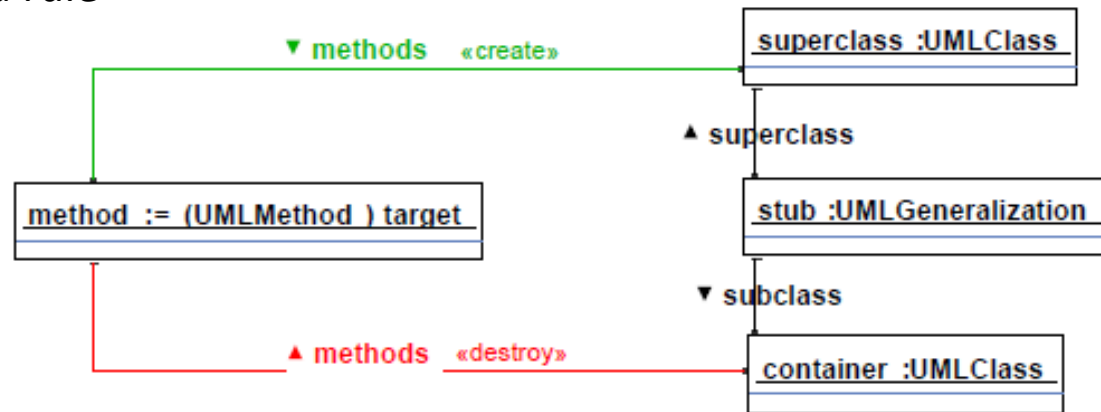
Elements annotated by addrFeature are included in the result iff addrFeature is selected during configuration

Model-to-model (II)

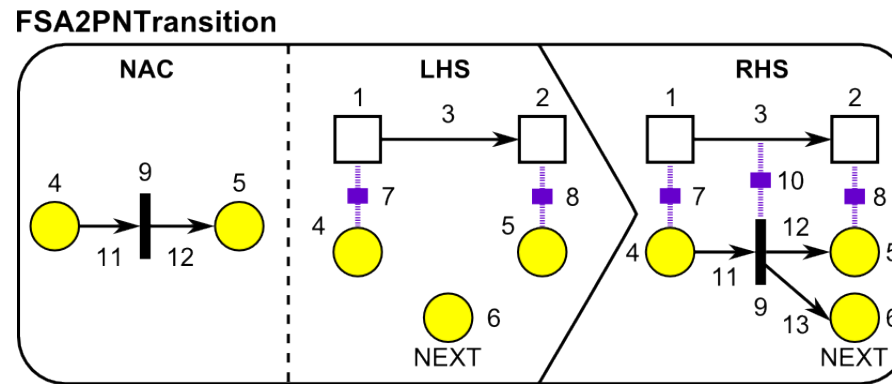
- Graph-transformation-based
 - Rule-based, in-place
 - graph patterns with elements to be created, deleted, left; possibly separated in RHS and LHS
 - Rule scheduling: predetermined (e.g., fixpoint) or programmable
 - Rich theory exists
 - Examples: VIATRA, Fujaba, AGG, ATOM3, GReAT, MOFLON (TGG)

Graph Trafo Rules

Fujaba rule



MoTif rule



VIATRA

```
gtrule liftAttrsR(inout CP, inout CS, inout A) =
{
  precondition pattern lhs(CP,CS,A,Par,Attr) =
  {
    Class(CP);
    Class.parent(Par,CS,CP);
    Class(CS);
    Class.attrs(Attr,CS,A);
    Attribute(A);
  }
  postcondition pattern rhs(CP,CS,A,Par,Attr,Attr2) =
  {
    Class(CP);
    Class.parent(Par,CS,CP);
    Class(CS);
    Class.attrs(Attr2,CP,A);
    Attribute(A);
  }
  action {
    print("Rule liftAttrR is
  }
}
```

//execution of a GT rule for one attribute of a class
//variables Class1 and Class2 must be bound
choose A **apply** liftAttrsR(Class1,Class2,A);

//calling the rule for all attributes of a class
//variables Class1 and Class2 must be bound
forall A **do** **apply** liftAttrsR(Class1,Class2,A);

//calling the rule for all possible matches in parallel
forall C1, C2, A **do** **apply** liftAttrsR(C1,C2,A);
//Apply a GT rule as long as possible for the entire model
iterate
 choose C1, C2, A **apply** liftAttrsR(C1,C2,A)

Model-to-model (III)

- Relational
 - Declarative relation between models
 - Operational meaning derived (interpreted or generated)
 - Limited if general purpose; powerful if specialized
 - Examples: QVT Relational, MOFLON (TGG), Code Views

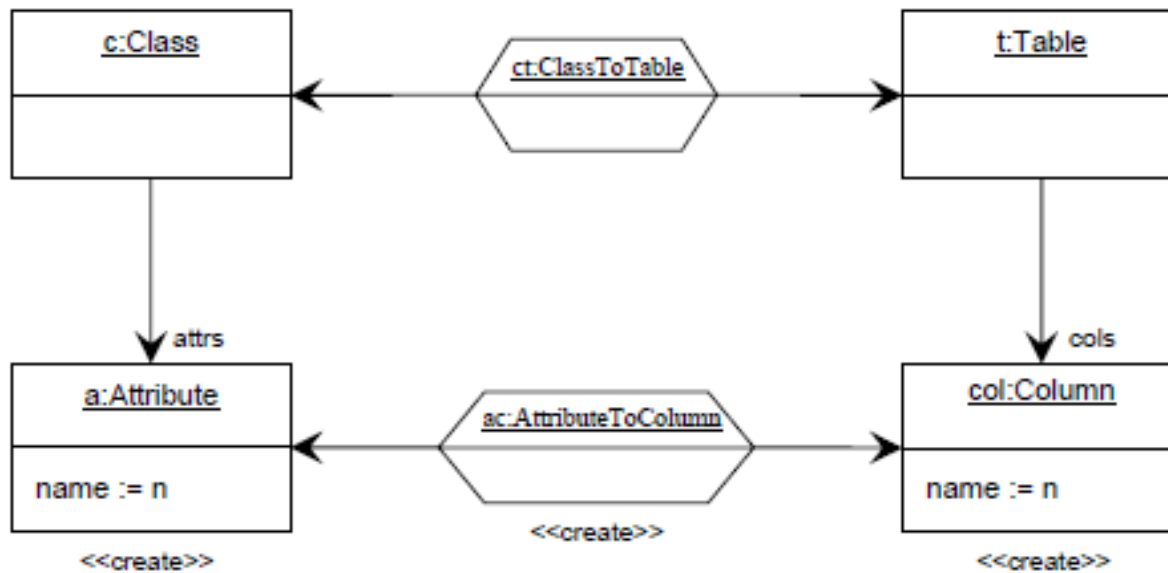
QVT Relational

```
top relation ClassToTable {
  domain uml c:Class {
    package = p:Package{},
    isPersistent = true,
    name = cn
  }
  domain rdbms t:Table {
    schema = s:Schema{},
    name = cn,
    cols = cl:Column {
      name = cn + '_tid',
      type = 'NUMBER'},
    pkey = cl
  }
  when {
    PackageToSchema (p, s);
  }
  where {
    AttributeToColumn (c, t);
  }
}
```

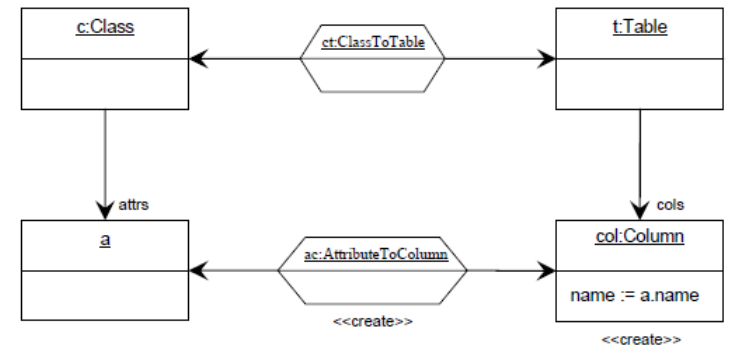
TGG

TGG operational rules

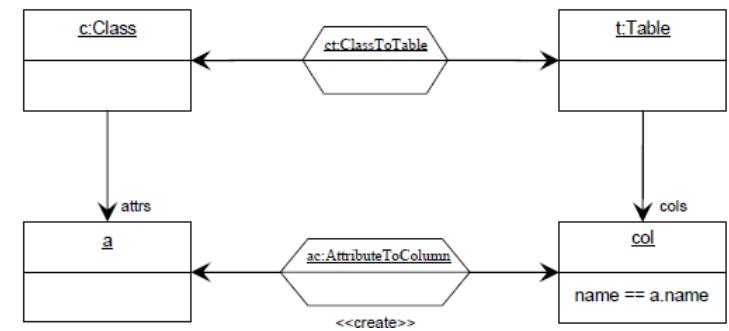
TGG rule



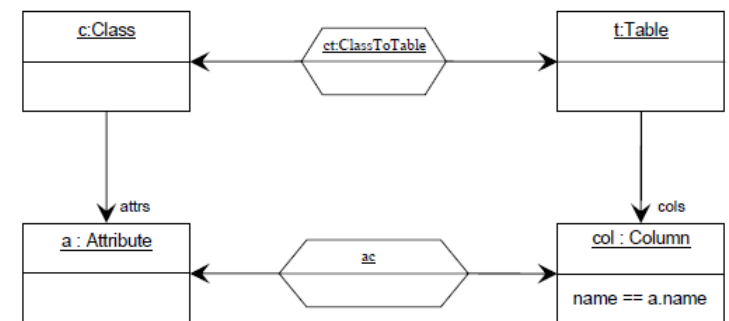
performForwardTransformation(a : Attribute)



performLinkCreation(a : Attribute, col : Column)



performConsistencyCheck(ac : AttributeToColumn)



Model-to-model (IV)

- Hybrid
 - Combination of different styles
 - Example: ATL combines rules with operational style
 - Many systems become increasingly hybrid as adapted to new problems
 - e.g., VIATRA

ATL

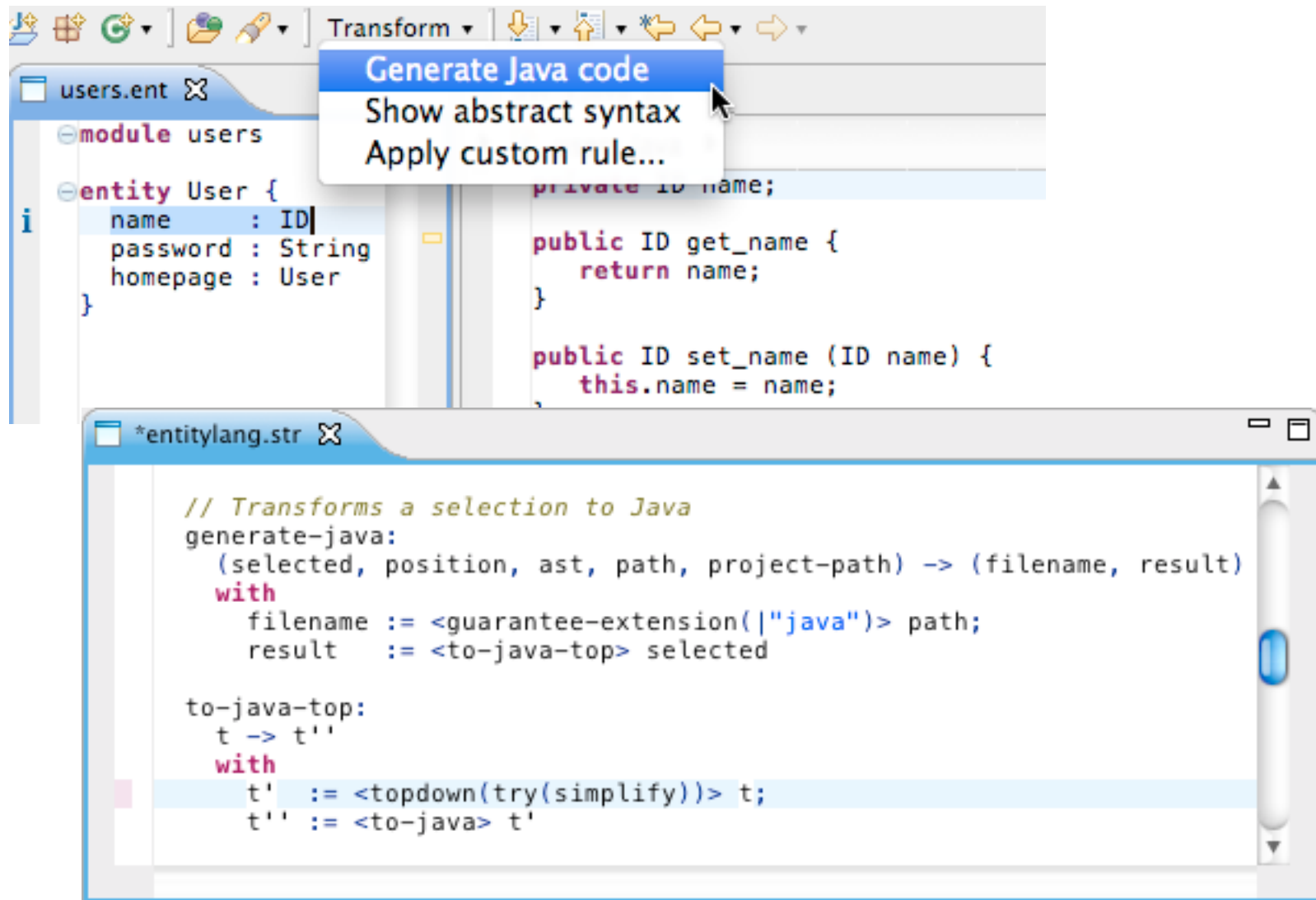
```
helper context Families!Member def: familyName: String =  
    if not self.familyFather.oclIsUndefined() then  
        self.familyFather.lastName  
    else  
        if not self.familyMother.oclIsUndefined() then  
            self.familyMother.lastName  
        else  
            if not self.familySon.oclIsUndefined() then  
                self.familySon.lastName  
            else  
                self.familyDaughter.lastName  
            endif  
        endif  
    endif;  
endif;
```

```
rule Member2Male {  
    from  
        s: Families!Member (not s.isFemale())  
    to  
        t: Persons!Male (  
            fullName <- s.firstName + ' ' + s.familyName  
        )  
}
```

Model-to-model (V)

- Other
 - Logic-programming based (e.g., Tefkat)
 - Can use other types of trafo systems as well
 - Program trafo systems: Spoofax (Stratego), TXL
 - MPS (AST-based)
 - XSLT

Spoofax (Stratego)



MPS

The screenshot displays the JetBrains MPS IDE interface. The main editor window shows the definition of the `Entity` concept, which extends `BaseConcept` and implements `INamedConcept`. It includes properties, children, references, concept properties, concept links, and concept property declarations. The `children` section is currently selected.

Overlaid on the main editor is a smaller window titled `Entity_Editor`, which is configured for the `Entity` concept. This window shows the `node cell layout` and the `inspected cell layout`.

The `node cell layout` is defined as follows:

```
[-  
  entity { name } {  
    (> % attributes % < )  
    /empty cell: <default>  
  }  
-]
```

The `inspected cell layout` is defined as:

```
<choose cell model>
```

The bottom of the IDE shows a tabbed interface with the following tabs: `Structure`, `Editor`, `Constraints`, `Behavior`, `Typesystem`, `Actions`, `Refactorings`, and `Intentions`. The `Inspector` panel is also visible, showing the `jetbrains.mps.lang.editor.structure.CellModel_RefNodeList` with the following properties:

Property	Value
separator style	default
reverse order	false
element factory	(scope, node) -> node < > {

DSLs in MPS

```
function block Stuff on the Wings
  uses Environment, Aircraft, Fundamental Stuff
```

Aus dem Staudruck p_{dyn} lässt sich dann der aktuelle Auftrieb F_A mit c_a beschrieben und die Fläche durch A .

```
F_A : double [N] = p_dyn * A * c_a
```

```
[ c_a=0.3 p_dyn=61.25 A=10 -> 183.75
  p_dyn=61.25 A=10 c_a=0.6 -> 367.5 ]
```

Auch der Widerstand F_W berechnet sich ebenfalls.

```
F_W : double [N] = p_dyn * A * c_w
```

Angenommen wir haben mehrere Flügel n_{wings} .

```
F_A_tot : double [N] =  $\sum_{i \text{ in } n_{wings}} F_A$ 
```

```
[import blocks Environment
    Aircraft
    Stuff on the Wings]

public class TestClass extends <none> implements <none> {
    <<static fields>>

    <<static initializer>>
    <<fields>>
    <<properties>>
    <<initializer>>
    public TestClass() {
        <no statements>
    }

    public void calcLift() {
        values air = (| Environment.rho = 1.225 |);
        values planeStatic = (| Aircraft.A = 10.0, Aircraft.c_a = 0.5, Aircraft.c_w = 0.05 |);
        double lift = Stuff on the Wings.F_A (air, planeStatic);
        System.err.println(lift);
    }

    public static void main(string[] args) {
        new TestClass().calcLift();
    }
}
```

Comparing model trafos in action

- Graph Transformation Tool Contest
 - See Arend Rensink and Pieter Van Gorp. Graph Transformation Tool Contest 2008. International Journal on Software Tools for Technology Transfer (DOI: 10.1007/s10009-010-0157-7)
 - Pieter Van Gorp and Rik Eshuis. Transforming Process Models: executable rewrite rules versus a formalized Java program. Applications track of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2010, Oslo). Lecture Notes in Computer Science, 2010, Volume 6395, pp. 258-272

Summary

- Distinction between program and model trafo fuzzy
- Great diversity of applications
- Great number and diversity of systems and languages
- Mostly academic, but situation improves
 - Spin-offs from academia
 - Still no prominent model trafo tool from a major vendor

References

- For an extensive bibliography in model trafoes see
 - <http://msdl.cs.mcgill.ca/people/eugene/30> literature
 - and the IBM Systems Journal 2006 survey at <http://gsd.uwaterloo.ca/node/68>

Acknowledgments

- Thanks to
 - Eugene Syriani and Hans Vangheluwe for allowing me to reuse some of the sample system screenshots from their DSM-TP2010 model trafo tutorial
 - Istwan Rath from the VIATRA team for discussing with me the ins and outs of graph transformation systems
 - Markus Voelter for allowing me to reuse the MPS screenshots
- The lions share of the material is based on the model trafo survey with Simon Helsen that appeared in the IBM Systems Journal in 2006

Questions?