5. Model Transformation

Juan de Lara, Elena Gómez, Esther Guerra
{Juan.deLara, MariaElena.Gomez, Esther.Guerra}@uam.es
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Indice

- **Introduction.** 
- Introduction to Graph Transformation.
- Advanced Graph Transformation
- ETL.
- QVT.
- Other Languages.
- Applications.
- Bibliography.

Introduction

- In Model-Driven Development, models are the core asset.
- Model-centric instead of code-centric.
- Models are used to: design, simulate, optimize, generate code and test the final application.

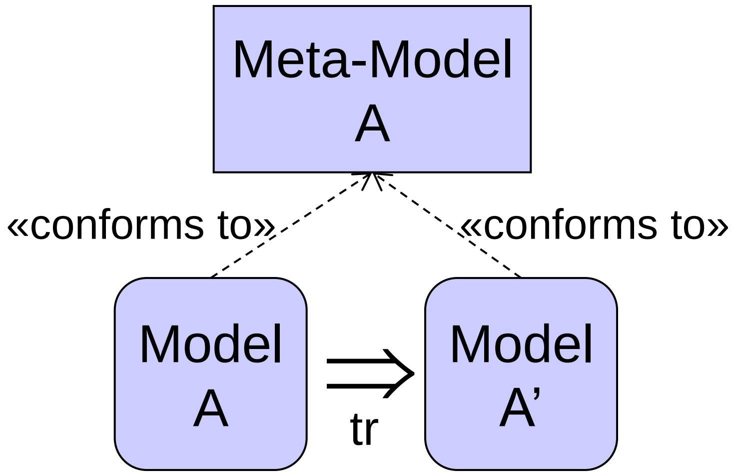
- Need for model manipulations!

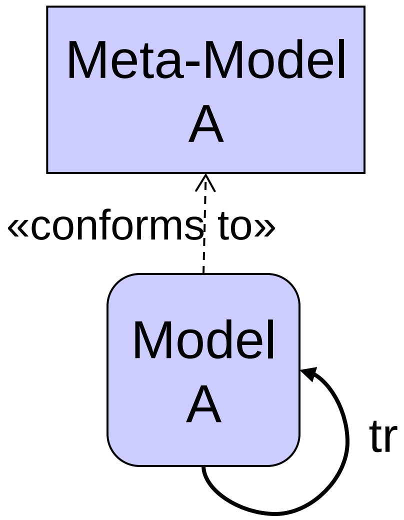
Types of Transformations

Domain	Level of Abstraction: Horizontal	Level of Abstraction: Vertical
Inter-Formalism (exogenous)	Analysis, PIM-PIM, Migration	Simulation, PIM-PSM
Intra-Formalism (endogenous)	Optimisation, Re-designs, Re-factoring, Animation	Abstraction, Refinement

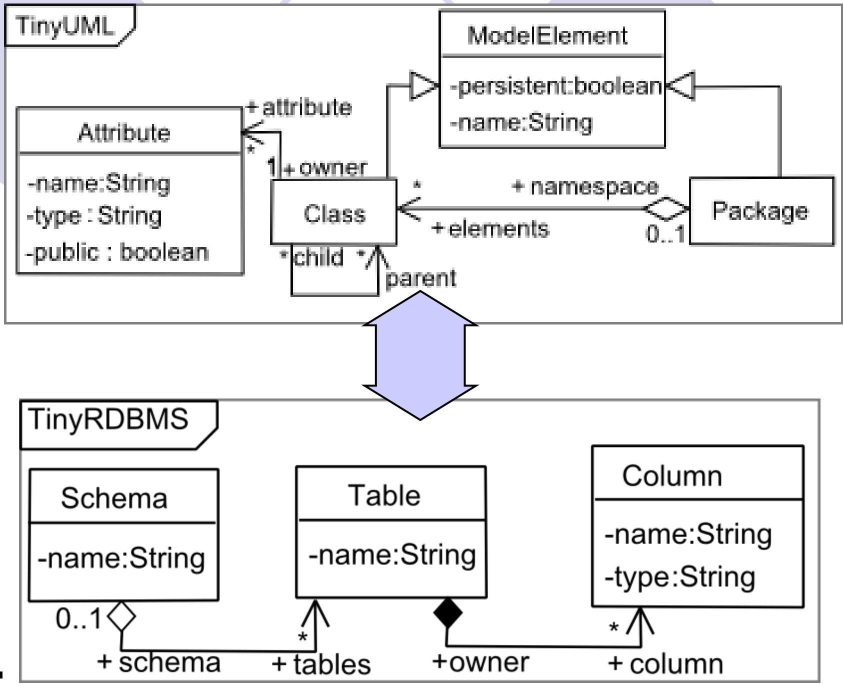
Intra-formalism

- In-place (or not).
- Optimization.
- Refactoring.
- Simulation or animation.





Model-to-Model

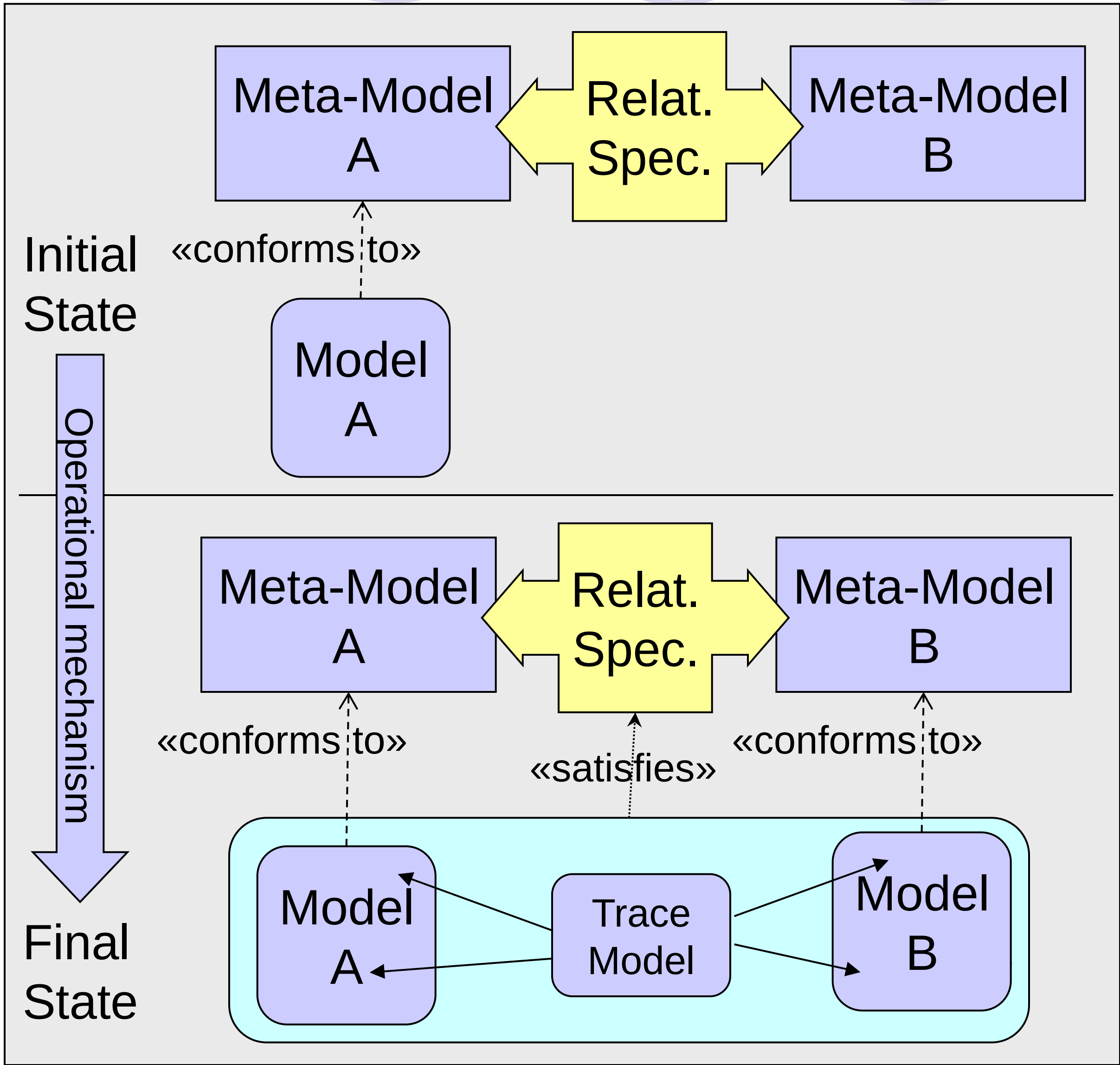


- Transform a model from a source to a target meta-model.
- Use domain specific languages to describe such transformation.
  - **Operational** languages. Two programs to transform source-to-target and target-to-source.
  - **Declarative**/Bidirectional. One specification can describe the transformation in both directions.
    - Compilation into operational mechanisms.

Model-to-Model Transformations

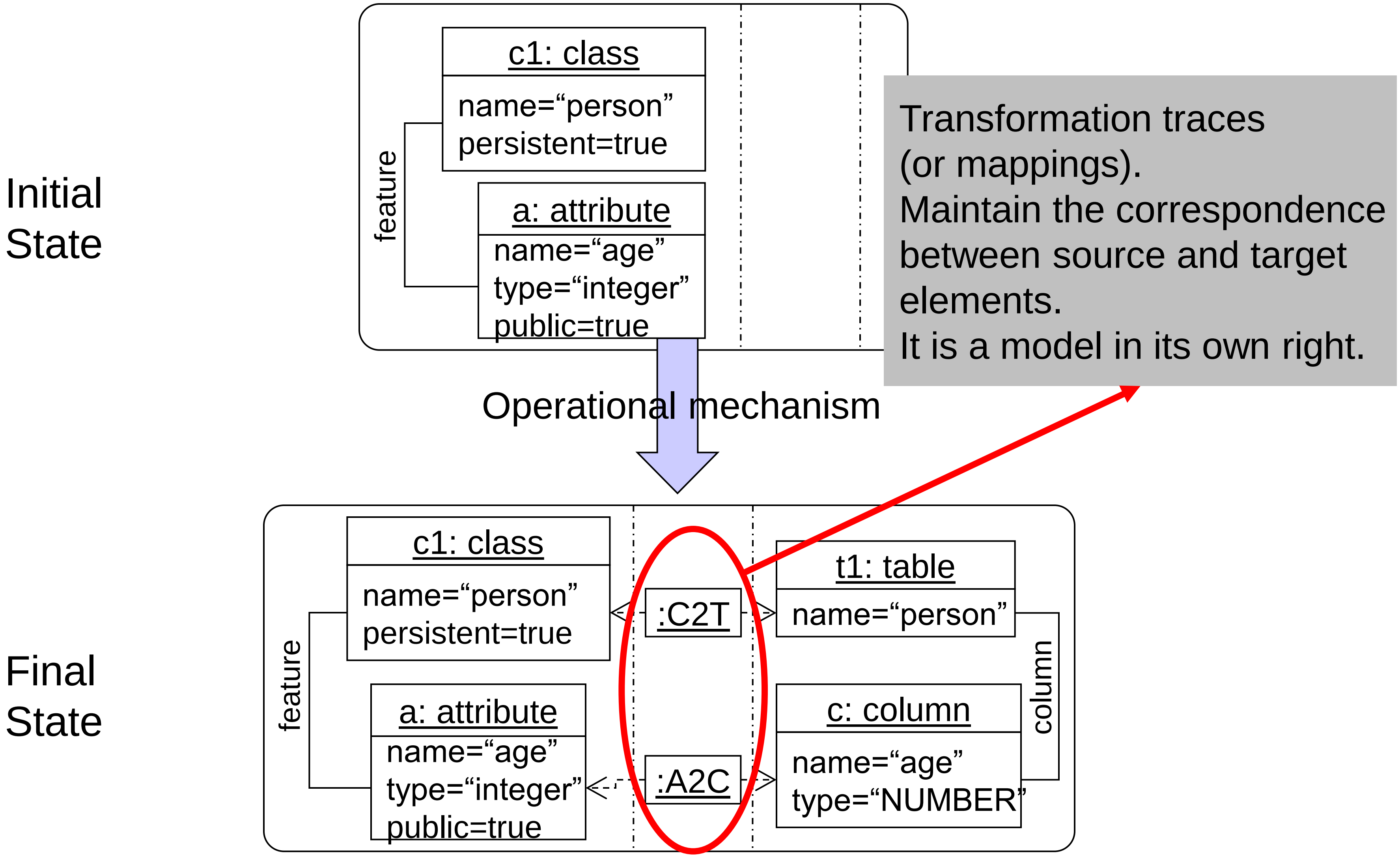
Batch Transformations

- Generate model B from model A.
- Model B is empty at the beginning.
- There can be more than one model B that together with A satisfies the specification.
- Target-to-Source is the symmetrical situation



Model-to-Model Transformations

Batch Transformation (forward)

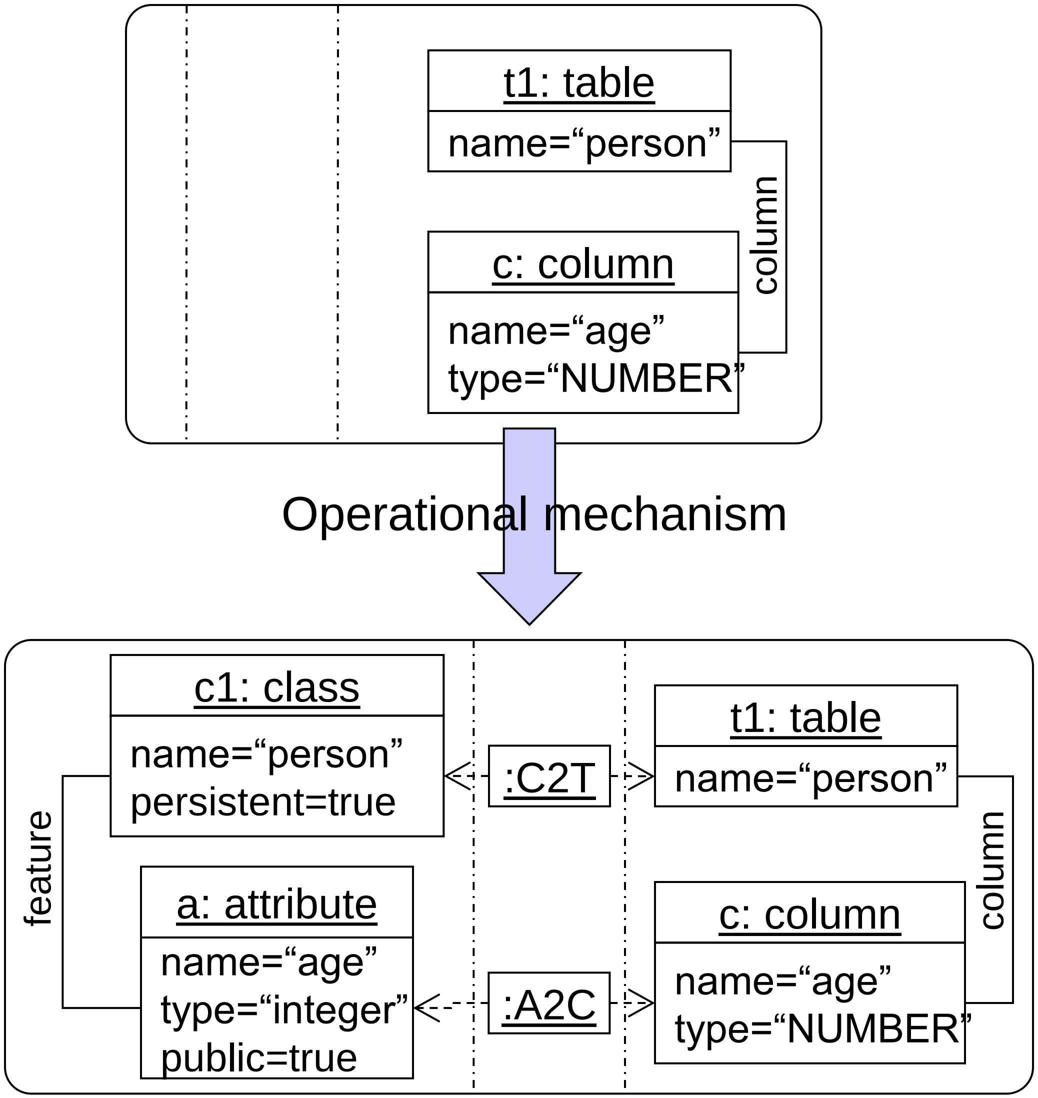


Operational Scenarios

Batch Transformation (backwards)

Initial State

Final State



Types of Languages

- Visual vs. Textual.
  - **Visual:** Graph grammars, QVT, MOLA, VIATRA, GREAT.
  - **Textual:** QVT, ATL, ETL, RubyTL, VIATRA, term-rewriting (MAUDE).

- Formal vs. Semi-formal.
  - **Formal:** Graph grammars, term-rewriting, VIATRA.
  - **Semi-formal:** QVT, ATL, ETL, RubyTL, MOLA, GREAT.

- Declarative, Imperative, Hybrid.
  - **Declarative:** Graph grammars, term rewriting.
  - **Imperative:** MOLA.
  - **Hybrid:** QVT, ATL, ETL, RubyTL, GREAT, VIATRA.

- Inter-formalism, intra-formalism, both.
  - **Inter-…:** Triple graph grammars, QVT, RubyTL, ATL, ETL.
  - **Intra-…:** Graph grammars, VIATRA, GREAT.
  - **Both:** Graph grammars, VIATRA.

Indice

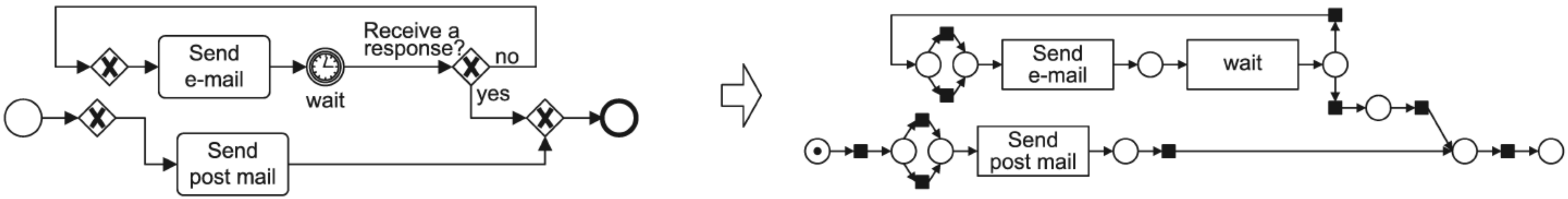
- Introduction.
- **Introduction to Graph Transformation.** 
- Advanced Graph Transformation
- ETL.
- QVT.
- Other Languages.
- Applications.
- Bibliography.

Indice

- Introduction.
- Introduction to Graph Transformation.
- Advanced Graph Transformation.
- **Model to Model Transformations.** 
- M2M Transformation Languages
- Applications.
- Bibliography.

Some examples

- Process models to Petri nets

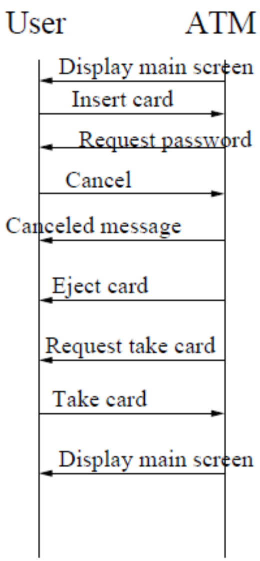


- Input: A BPMN process model
- Output: A Petri net
- Purpose: Analysis of the BPMN process model by analysing the Petri net instead.

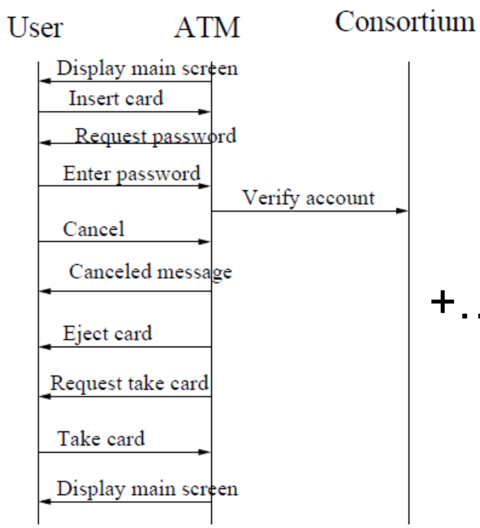
Dijkman, Dumas, Ouyang. Semantics and analysis of business process models in BPMN. Information and Software Technology, 50:12, pp.:1281-1294 (2008)

Some examples

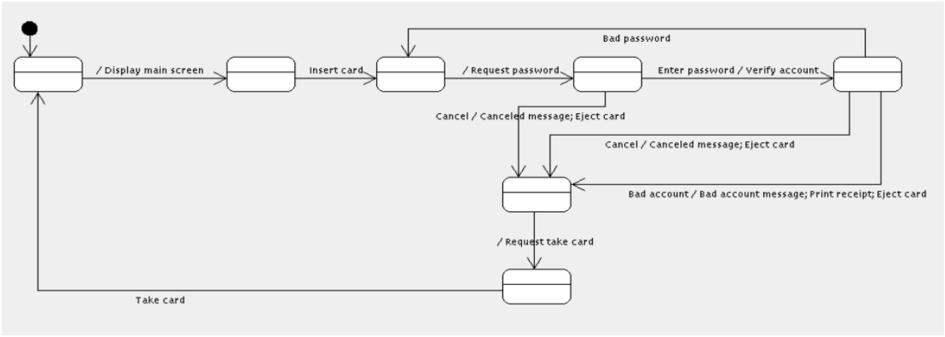
- Sequence diagrams to Statecharts



+



+…=

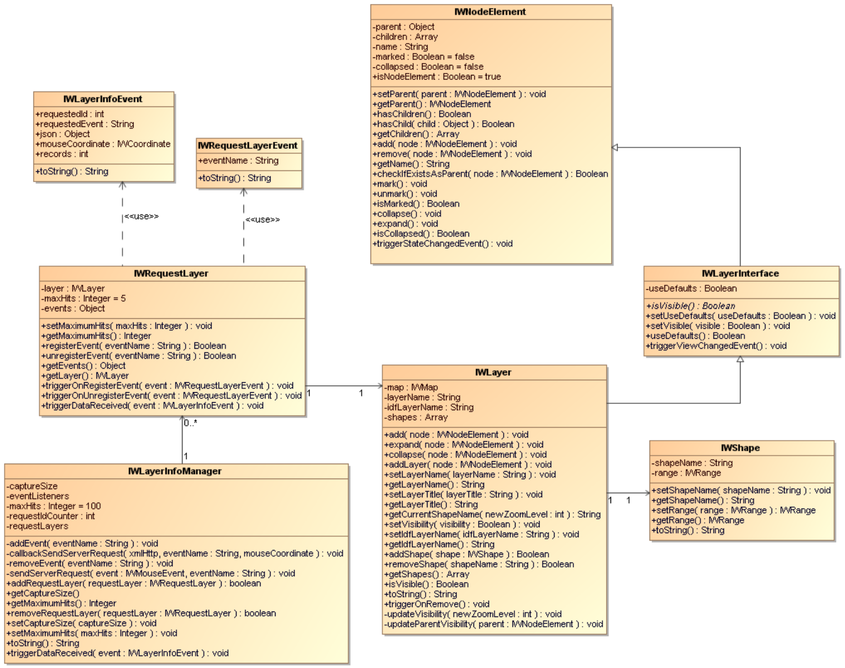


- Input: Set of sequence diagrams.
- Output: A Statechart.
- Purpose: Transition from analysis to design, reverse engineering.

Whittle, Schumann. "Generating Statecharts Designs from Scenarios". ICSE'2000.

Some examples

- Class Diagrams to Metrics:



Metric	Value
#Class	8
DIT	2
…	…

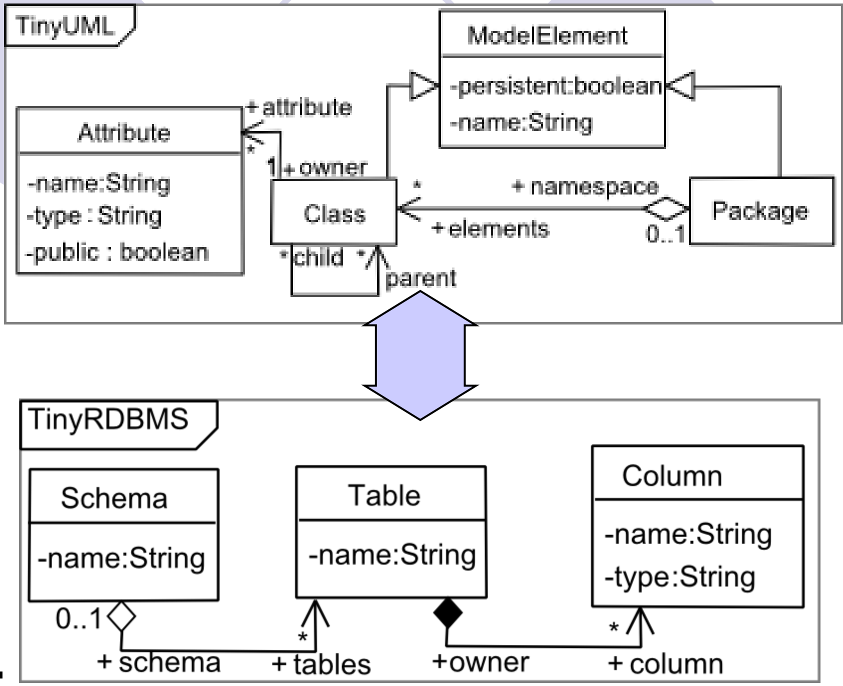
- Input: A Class Diagram.
- Output: A model showing complexity metrics (e.g., number of classes, max depth of inheritace tree, etc).
- Purpose: Quality Assurance

ATL Zoo: http://www.eclipse.org/atl/atlTransformations/

Scenarios

- Batch:
  - A complete target model is created from a source one.
- Incremental:
  - The target model is updated upon changes in the source.
- Bidirectional:
  - Both source/target models are modified.

Model-to-Model

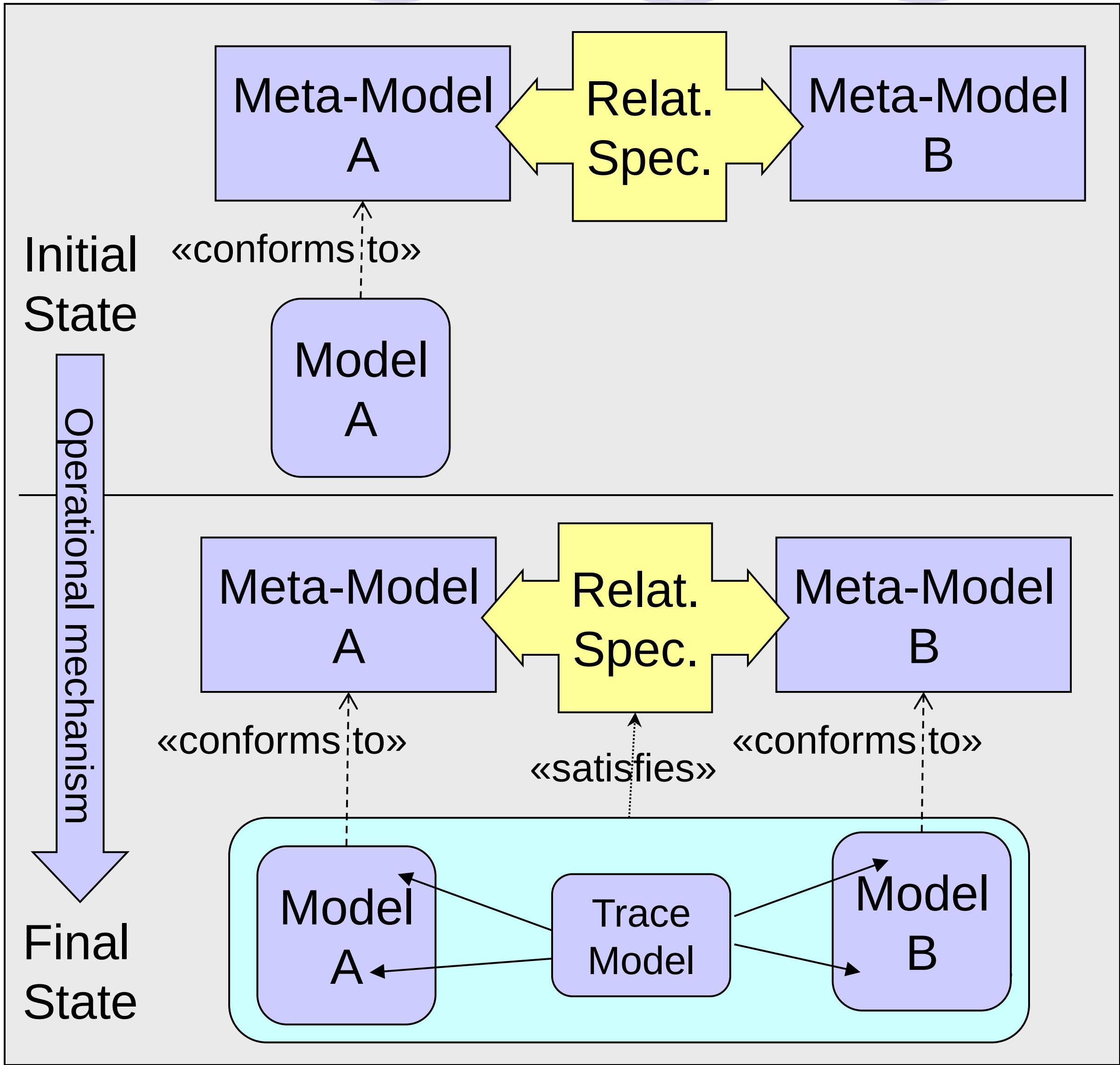


- Transform a model from a source to a target meta-model.
- Use domain specific languages to describe such transformation.
  - **Operational** languages. Two programs to transform source-to-target and target-to-source.
  - **Declarative**/Bidirectional. One specification can describe the transformation in both directions.
    - Compilation into operational mechanisms.

Model-to-Model Transformations

Batch Transformations

- Generate model B from model A.
- Model B is empty at the beginning.
- There can be more than one model B that together with A satisfies the specification.
- Target-to-Source is the symmetrical situation

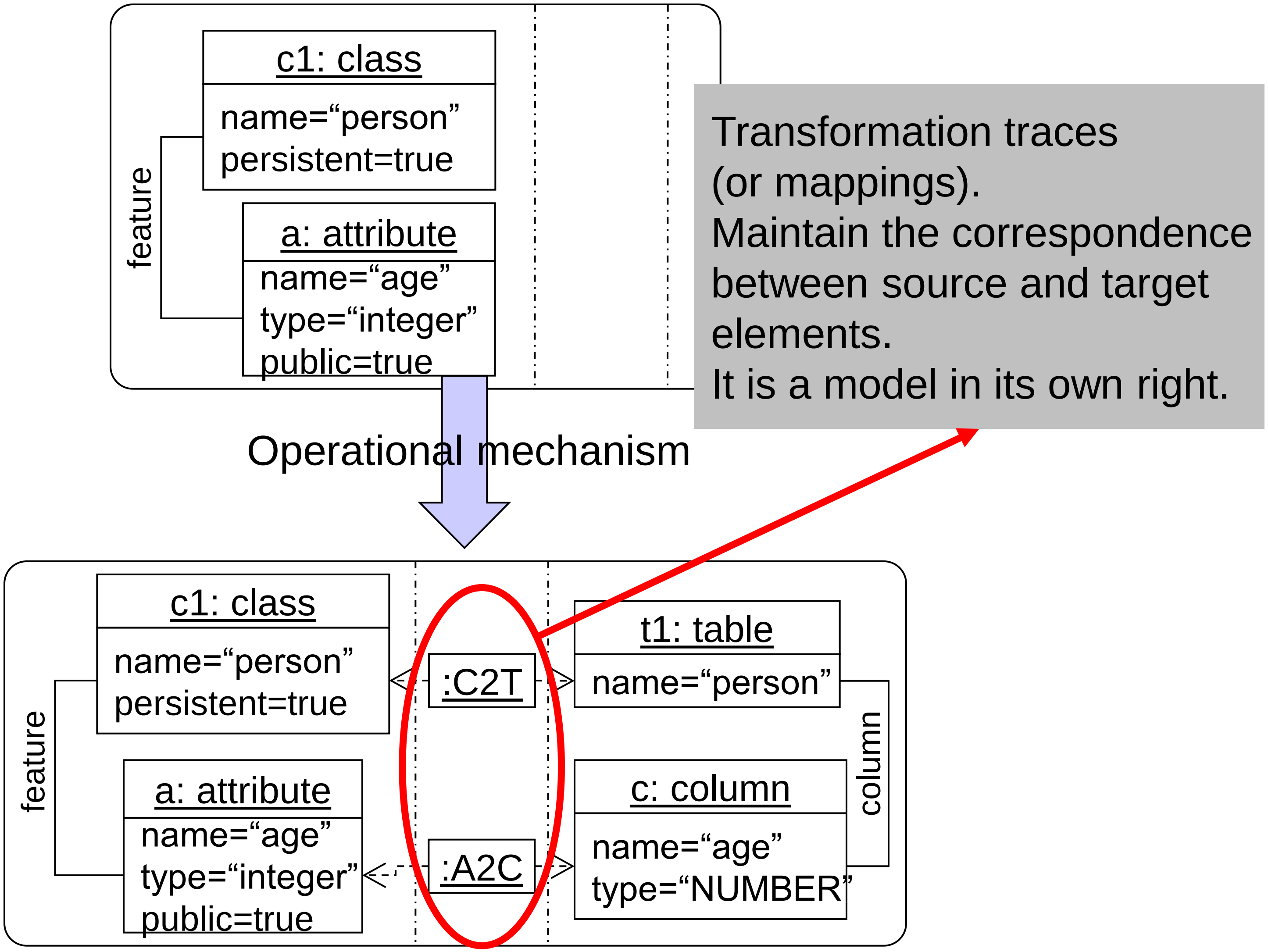


Model-to-Model Transformations

Batch Transformation (forwards)

Initial State

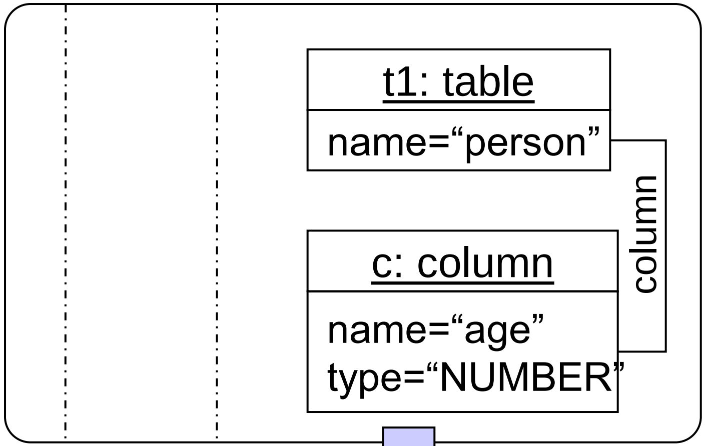
Final State



Operational Scenarios

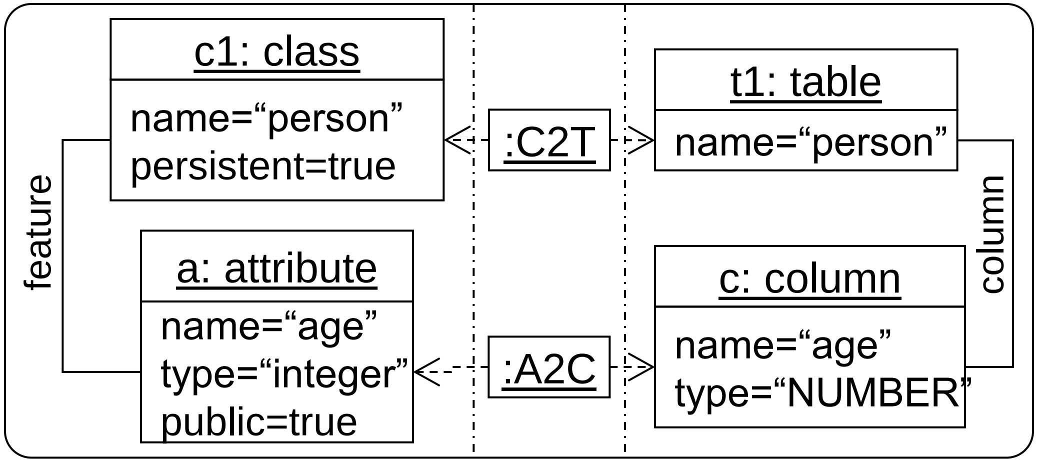
Batch Transformation (backwards)

Initial State



Operational mechanism

Final State



Types of Languages

- Visual vs. Textual.
  - **Visual:** Graph grammars, QVT, MOLA, VIATRA, GREAT.
  - **Textual:** QVT, ATL, ETL, RubyTL, VIATRA, term-rewriting (MAUDE).

- Formal vs. Semi-formal.
  - **Formal:** Graph grammars, term-rewriting, VIATRA.
  - **Semi-formal:** QVT, ATL, ETL, RubyTL, MOLA, GREAT.

- Declarative, Imperative, Hybrid.
  - **Declarative:** Graph grammars, term rewriting.
  - **Imperative:** MOLA.
  - **Hybrid:** QVT, ATL, ETL, RubyTL, GREAT, VIATRA.

- Inter-formalism, intra-formalism, both.
  - **Inter-…:** Triple graph grammars, QVT, RubyTL, ATL, ETL.
  - **Intra-…:** Graph grammars, VIATRA, GREAT.
  - **Both:** Graph grammars, VIATRA.

Index

- Introduction.
- Introduction to Graph Transformation.
- Advanced Graph Transformation.
- Model to Model Transformations.
- **M2M Transformation Languages** 
  - ETL
  - QVT.
  - Triple Graph Grammars.
  - Other Languages.
- Bibliography.

ETL

- Part of the ***Epsilon*** family of languages for model management.
- Simple model-to-model transformation language (exogenous transformations).
- Support for an arbitrary number of input/output models.
- Rule-based, but with imperative code (EOL).

ETL

Source meta-model

```
package Tree;
class Tree {
  val Tree[*]#parent children;
  ref Tree#children parent;
  attr String label;
}
```

Target meta-model

```
package Graph;
class Graph {
  val Node[*] nodes;
}

class Node {
  attr String name;
  val Edge[*]#source outgoing;
  ref Edge[*]#target incoming;
}

class Edge {
  ref Node#outgoing source;
  ref Node#incoming target;
}
```

Transformation

```
rule Tree2Node
  transform t : Tree!Tree
  to n : Graph!Node {
    n.name := t.label;
    if (t.parent.isDefined()) {
      var e : new Graph!Edge;
      e.source ::= t.parent;
      e.target := n;
    }
}
```

Returns the Node object in which t.parent has been transformed

ETL

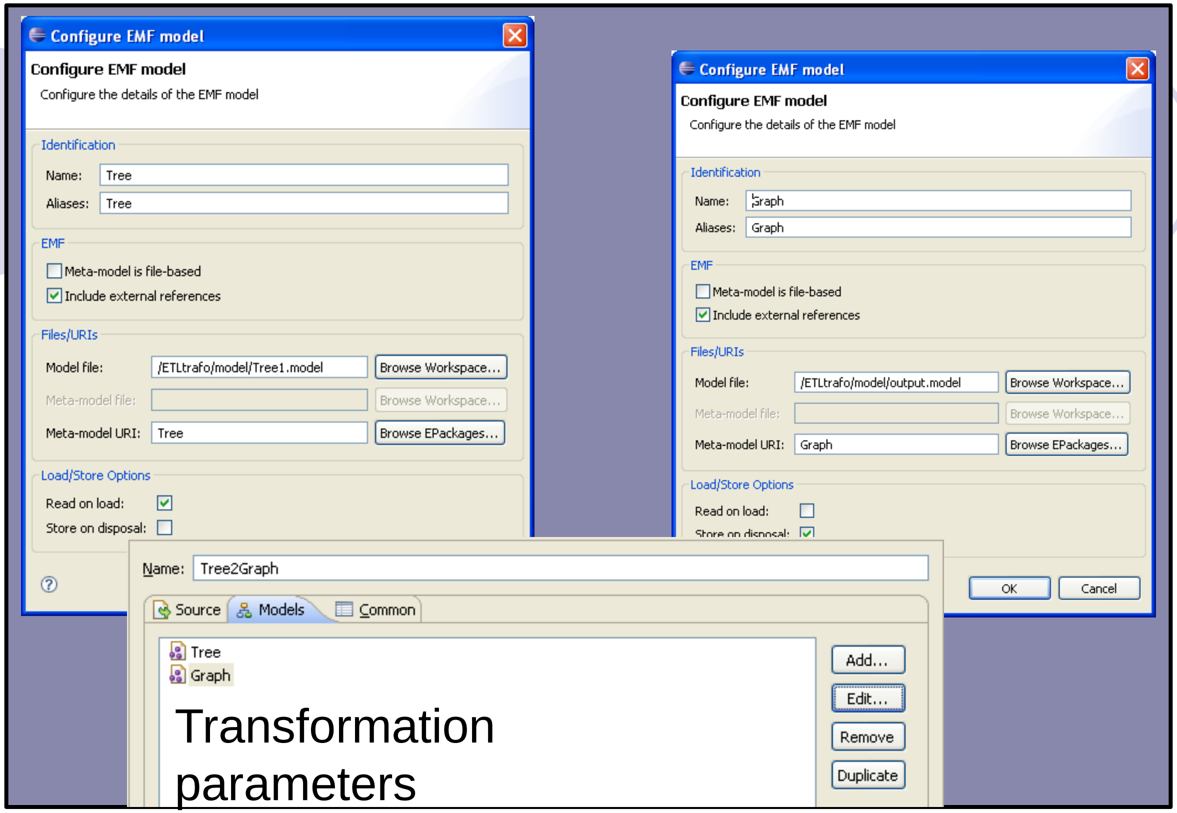
- Implicit creation of traces (correspondences between source and target elements).

- Each rule is executed at most once for each source element.

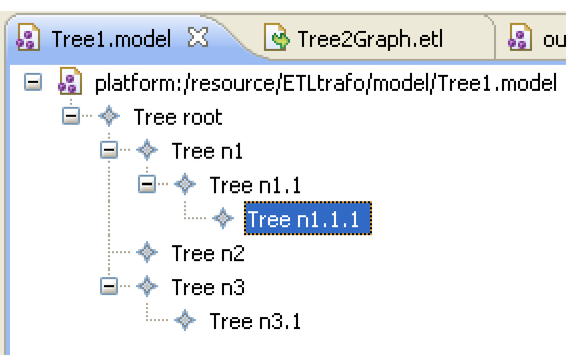
- Two kinds of rules:
  - ***Lazy***: needs to be called from another one (by means of ::= or *equivalent()*).

  - ***default***: no need to call them (the execution engine schedules them automatically).

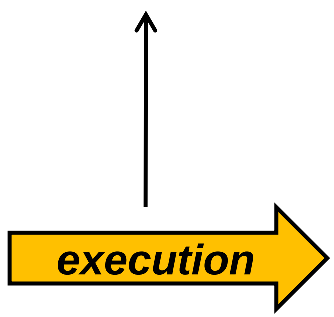
ETL



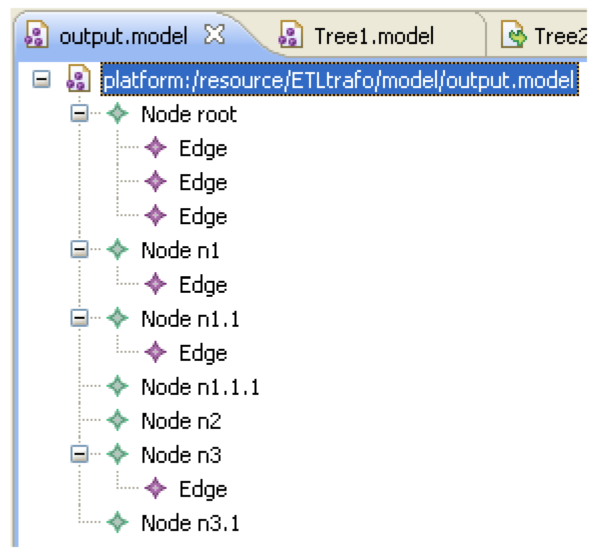
Configure EMF model
Configure the details of the EMF model
Identification
Name: Tree
Aliases: Tree
EMF
Meta-model is file-based
Include external references
Files/URIs
Model file: /ETLtrafo/model/Tree1.model
Browse Workspace...
Meta-model file:
Meta-model URI: Tree
Browse EPackages...
Load/Store Options
Read on load:
Store on disposal:
Configure EMF model
Configure the details of the EMF model
Identification
Name: Graph
Aliases: Graph
EMF
Meta-model is file-based
Include external references
Files/URIs
Model file: /ETLtrafo/model/output.model
Browse Workspace...
Meta-model file:
Meta-model URI: Graph
Browse EPackages...
Load/Store Options
Read on load:
OK
Cancel
Name: Tree2Graph
Source
Models
Common
Tree
Graph
Add...
Edit...
Remove
Duplicate
Transformation parameters



Tree1.model
Tree2Graph.etl
platform:/resource/ETLtrafo/model/Tree1.model
Tree root
Tree n1
Tree n1.1
Tree n1.1.1
Tree n2
Tree n3
Tree n3.1



execution



output.model
Tree1.model
platform:/resource/ETLtrafo/model/output.model
Node root
Edge
Edge
Edge
Node n1
Edge
Node n1.1
Edge
Node n1.1.1
Node n2
Node n3
Edge
Node n3.1

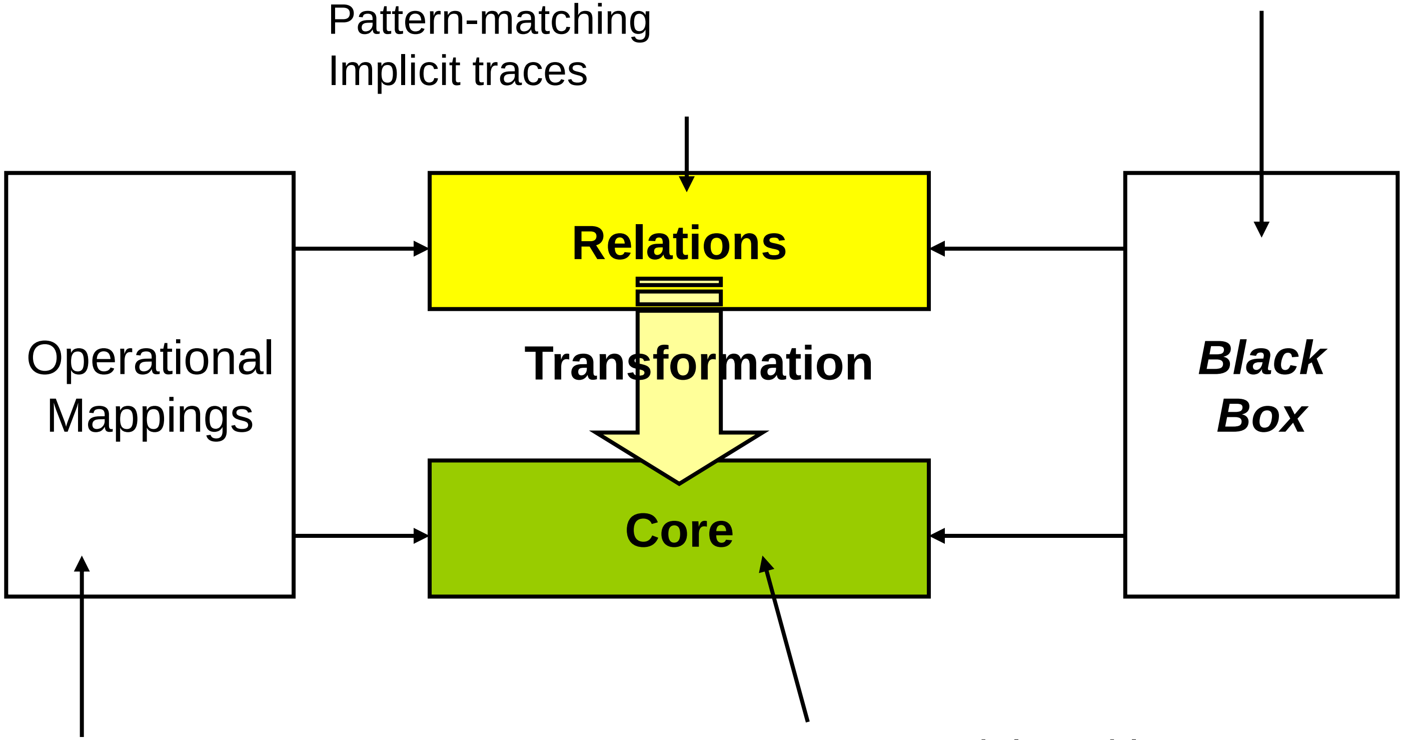
QVT

- QVT: Query/Views/Transformations.
- Language proposed by the OMG to express model transformations in the MDA.
- Model-to-Model transformations.
- http://www.omg.org/docs/ptc/05-11-01.pdf
- Three parts:
  - Relations.
  - Core.
  - Operational.

QVT.
Architecture.

Meta-model and
“user-friendly” language
Pattern-matching
Implicit traces

“plug-ins”
Domain-specific libraries to calculate attribute values.



Imperative.
Extensions to OCL with secondary effects

Meta-model and language
Minimal extensions to EMOF and OCL
Traces as MOF models.

QVT.
Use scenarios.

- “Check-only” transformations to verify that the models are related correctly, according to the transformation specification.
- Many-models to many-models transformations.
- Uni-directional transformations.
- Bi-directional transformations.
- Establish relations between pre-existent models.
- Incremental updates of a model.

QVT.
Relations Language

- Transformations:

```
Transformation <name> (        <src-model-name>: <src-metamodel>,
                               <tgt-model-name>: <tgt-metamodel>) {
… /* Relations… */
}
```

- One of the models is the source and the other the target.

- To check (“***checkonly***”) the consistency of the two models, or to modify one of them (“***enforce***”) and make them consistent.

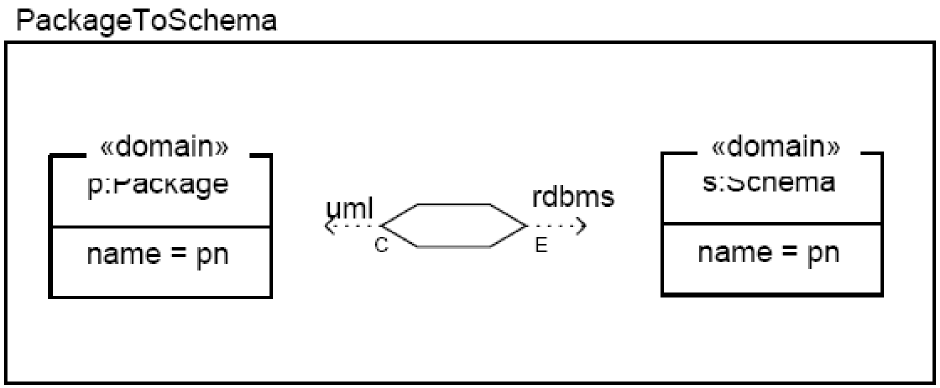
- If it is “***enforce***”, the model is modified to make the relations between them correct (create, erase or modify the “enforceable” target model).

QVT.

Relations and Domains.

- A relation declares relations that need to be satisfied between two models.
- Made of two domains + *when* and *where* clauses (control structures).
- Example:

Relation PackageToSchema
/* Map each package to a Schema */
{
domain uml p: Package{name=pn}
domain rdbms s: Schema{name=pn}
}

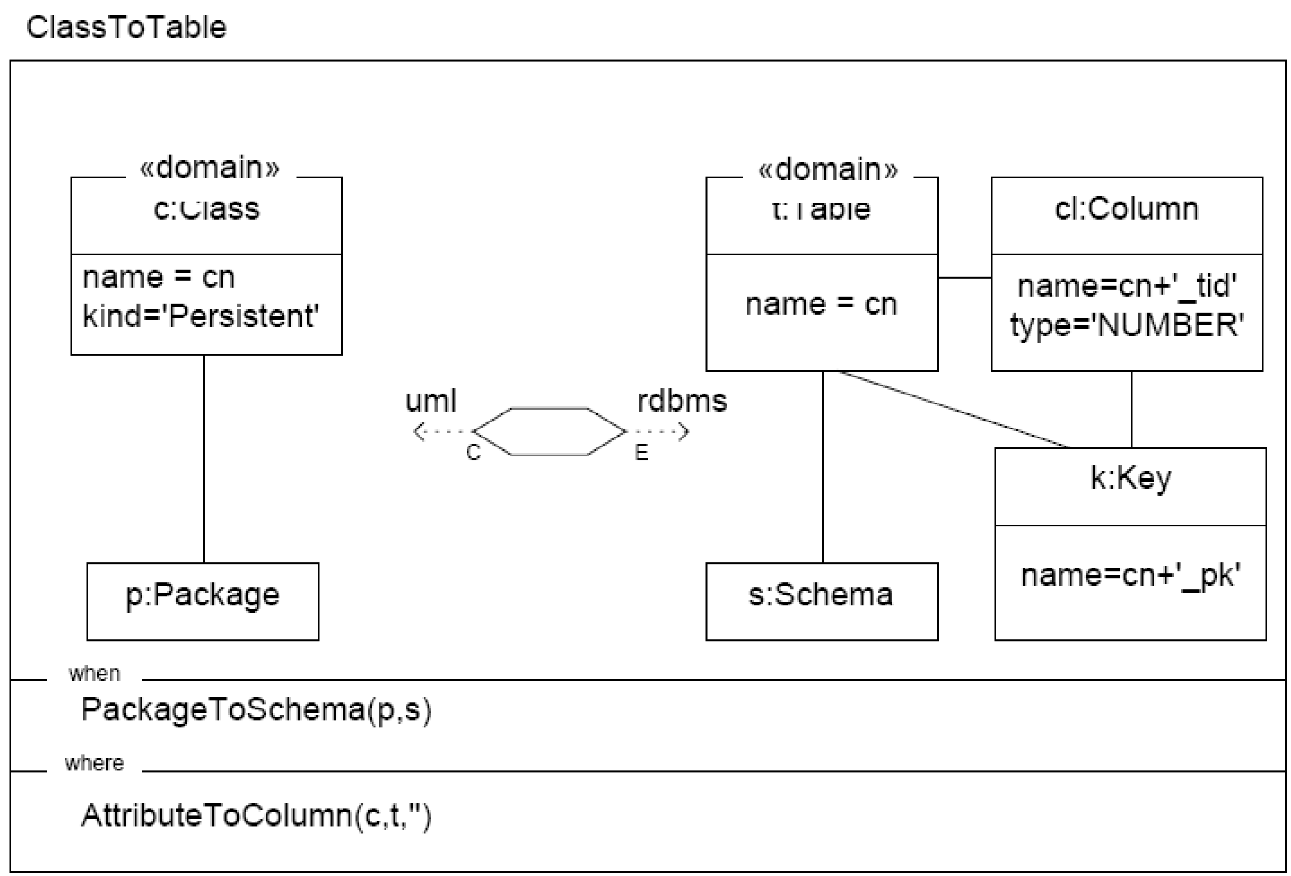


QVT.

When and Where clauses

- ***When***: Conditions in which the relation needs to be satisfied.
- ***Where***: Further restrictions that needs to be satisfied by some elements of the relation.

```
Relation ClassToTable
{
domain uml c:Class{
  namespace = p:Package {},
  kind='Persistent',
  name=cn
}
domain rdbms t:Table{
  schema = s: Schema{},
  name = cn,
  column = cl: Column{
    name = cn+'id'
    type = 'NUMBER'},
  primaryKey = k: PrimaryKey{
    name = cn+"_pk",
    column = cl}
}
When {PackageToSchema(p, s);}
Where {AttributeToColumn(c,t);}
}
```



QVT.

“Top-Level” relations

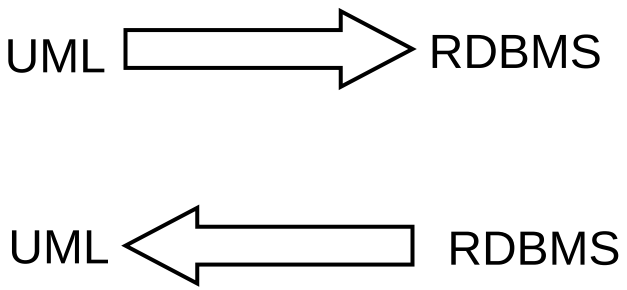
- Two kinds of relations: top-level and non top-level.
- The execution of a transformation requires all top-level relations to be satisfied.
- Of the non-top-level ones, only those invoked directly or indirectly from ***where*** clauses.
- Example:

```
Transformation umlRdbms (    uml: SimpleUML, rdbms: SimpleRDBMS ) {
    top relation PackageToSchema { … }
    top relation ClassToTable {…}
    relation AttributeToColumn {…}
}
```

QVT.
“Check” and “Enforce”

- Domains can be marked with “checkonly” or “enforce”.
- If we execute in the direction of a “checkonly” domain: if is verified if the transformation relations are satisfied.
- If we execute in the direction of an “enforced” domain: the model in the target domain is modified to satisfy the relations.
- Example:

<u>Scenario</u>



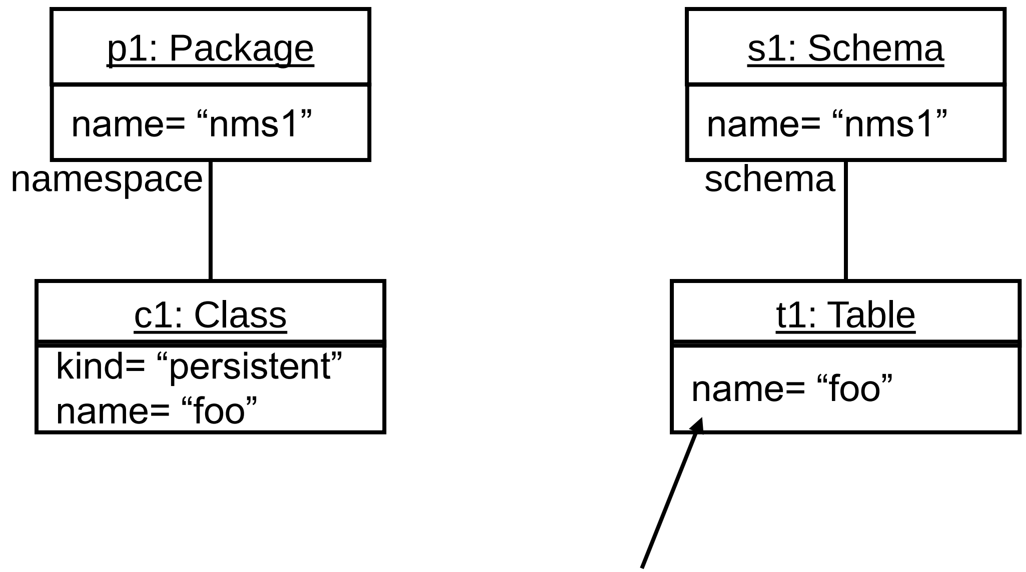
Relation PackageToSchema
/* Map each package to a Schema */
{
checkonly domain uml p: Package{name=pn}
enforce domain rdbms s: Schema{name=pn}
}

QVT.

Keys

- We can declare keys, which avoid duplicating objects when creating them.
- An object in an enforced domain is created if a match to its key does not exist.

```
Relation ClassToTable
{
domain uml c:Class{
  namespace = p:Package {},
  kind=‘Persistent’,
  name=cn
}
domain rdbms t:Table{
  schema = s: Schema{},
  name = cn,
  column = cl: Column{
    name = cn+’id’
    type = ‘NUMBER’},
  primaryKey = k: PrimaryKey{
    name = cn+”_pk”,
    column = cl}
}
Key Table{schema, name}
When {PackageToSchema(p, s);}
Where {AttributeToColumn(c,t);}
}
```

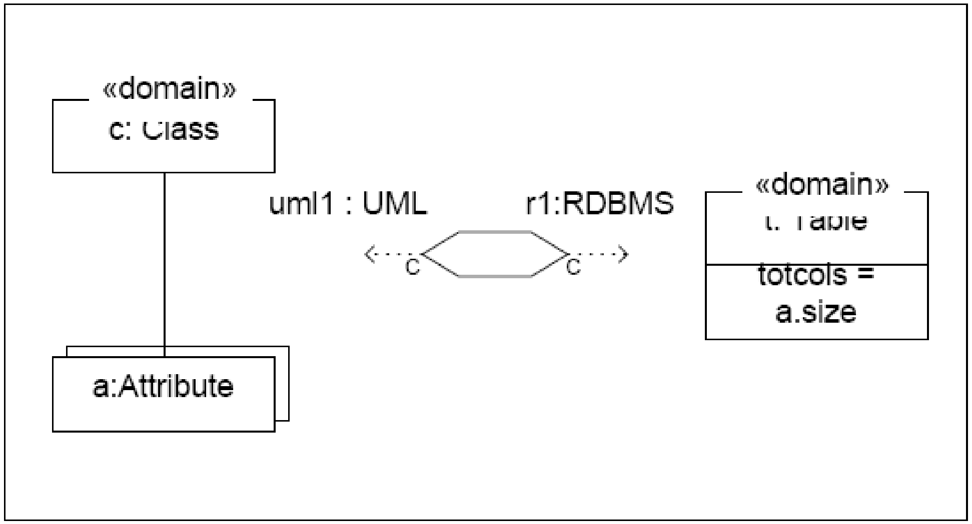


No need to create a new table:
its column attributes and primaryKey
need to be changed

QVT.

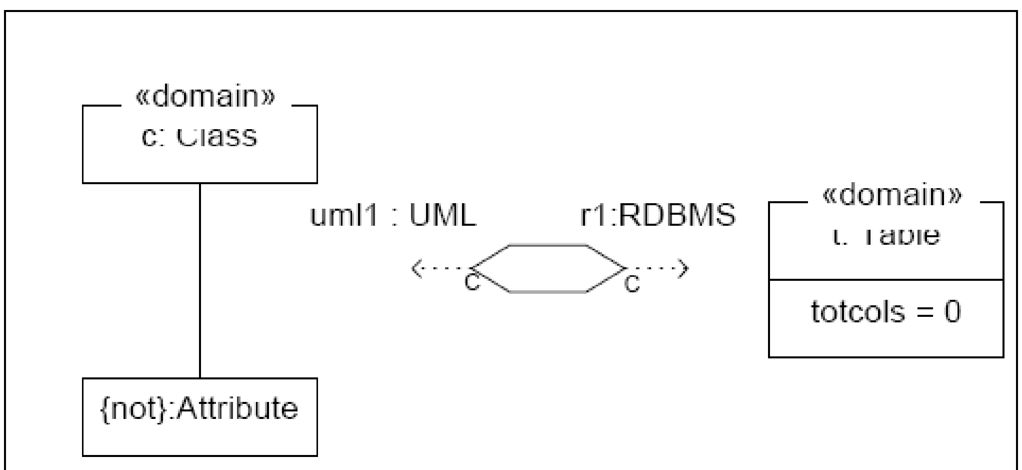
Sets

UML2Rel



Relation UML2Rel
{
Checkonly domain uml1 c:Class{
attribute = Set(Attribute){}
}
Checkonly domain r1 t:Table{
totcols=a.size
}

UML2Rel



Relation UML2Rel
{
Checkonly domain uml1 c:Class{
attribute = Set(Attribute){}
{attribute->size()==0}
}
Checkonly domain r1 t:Table{
totcols=0
}

Other Languages: ATL

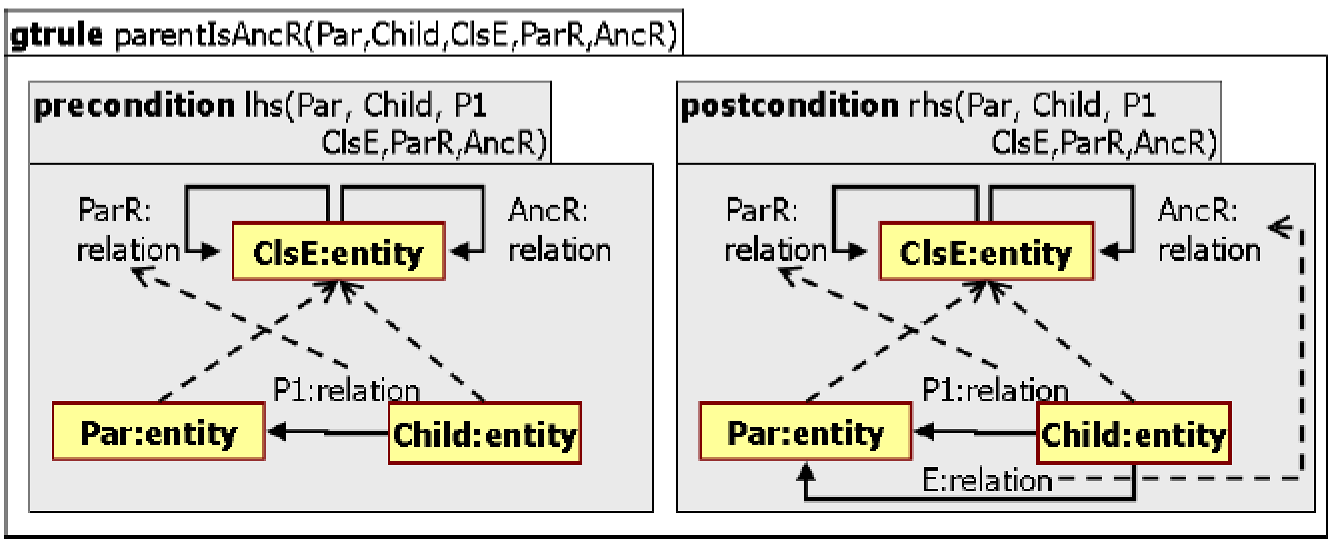
- ATLAS Transformation Language.
- Textual Language, declarative and imperative constructs.

```
rule PersistentClass2Table{
    from
        c : SimpleClass!Class (
            c.is_persistent and c.parent.oclIsUndefined()
        )
    to
        t : SimpleRDBMS!Table (
            name <- c.name
        )
}
```

Other Languages: VIATRA2

Graph-based:

- Nested negative patterns.
- Recursive patterns.
- Generic rules (type as arguments), rules that modify rules.



http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt-home/subprojects/VIATRA2/index.html

Bibliography

Graph Transformation:

- Handbook of Graph Grammars and Computing by Graph Transformation. 3 Vols. 1997. World Scientific.
- Ehrig, H., Ehrig, K., Prange, U., Taentzer, G. 2006. “Fundamentals of Algebraic Graph Transformation”.Springer.
- Ehrig, H., Ehrig, K., de Lara, J., Taentzer, G., Varró, D., Varró-Gyapay, S. “Termination Criteria for Model Transformation”. FASE 2005, LNCS 3442, pp.: 49-63.
- R. Bardohl, H. Ehrig, J. de Lara and G. Taentzer. “Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation”. FASE 2004, LNCS 2984, pp.: 214-228.

Triple Graph Grammars:

- http://www.es.tu-darmstadt.de/index2.php?page=2895
- Schürr, A.: "*Specification of Graph Translators with Triple Graph Grammars*", in: Tinhofer, G. (eds.): Springer-Verlag (pub.), Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science, Vol. 903, LNCS pp. 151-163, Heidelberg, June 1994
- Königs, A., Schürr, A. 2006. “*Tool Integration with Triple Graph Grammars: A Survey*”. Electronic Notes in Theoretical Computer Science. 148, pp.: 113-150.