Model Transformations

Transformation of Simplified UML Model
Into Simplified Rdbms Model Using
AToM^3 Meta-modeling Tool

Michal Antkiewicz
mantkiew@swen.uwaterloo.ca

Overview

- AToM^3 Meta-modeling tool
- Project description: UML to Rdbms transformation
- UML and Rdbms Meta-model
- AToM^3 Model Transformations
- Uml2Rdbms Transformation
- Demo
- AToM^3 Tool Classification
- AToM^3 Missing Features

AToM^3

- AToM^3: A Tool for Multi-formalism and Meta-Modeling
- Developed at the Modeling, Simulation and Design Lab in the School of Computer Science of McGill University
- Written entirely in Python
- Meta-models available with distribution:
  - Entity-Relationship
  - Deterministic and nondeterministic FSA
  - Petri Nets
  - Data Flow Diagrams
  - Structure Charts

AToM^3 (II)

- Uses Entity-Relationship formalism to describe models and meta-models
- Allows custom graphical representations of modeling concepts
- Performs model to model transformations in any modeling language which can be meta-modeled in Entity-Relationship
- Generates tools to visually manipulate models described in given meta-model and performs transformations on them
- Transformations are realized by pattern matching and graph rewriting

Project: Uml to Rdbms transformations

1. Each class in the simplified UML model with kind = "persistent" is mapped on to a table:
   a) Table name = “t_” + Class name
   b) String attributes are mapped into VARCHAR columns
   c) Integer attributes are mapped into NUMBER columns
   d) Class attributes are not mapped into columns. Instead, columns are added for each primitive type attribute of a class recursively. Column name = class attribute name + “_” + primitive attribute name.
   e) Attributes of kind = “primary” are collected in Primary Key of table

Project: Uml to Rdbms transformations (II)

2. Directed association between two persistent classes is mapped into foreign key:

   a) For each attribute with kind = “primary” in destination class, column is added to table associated with source class

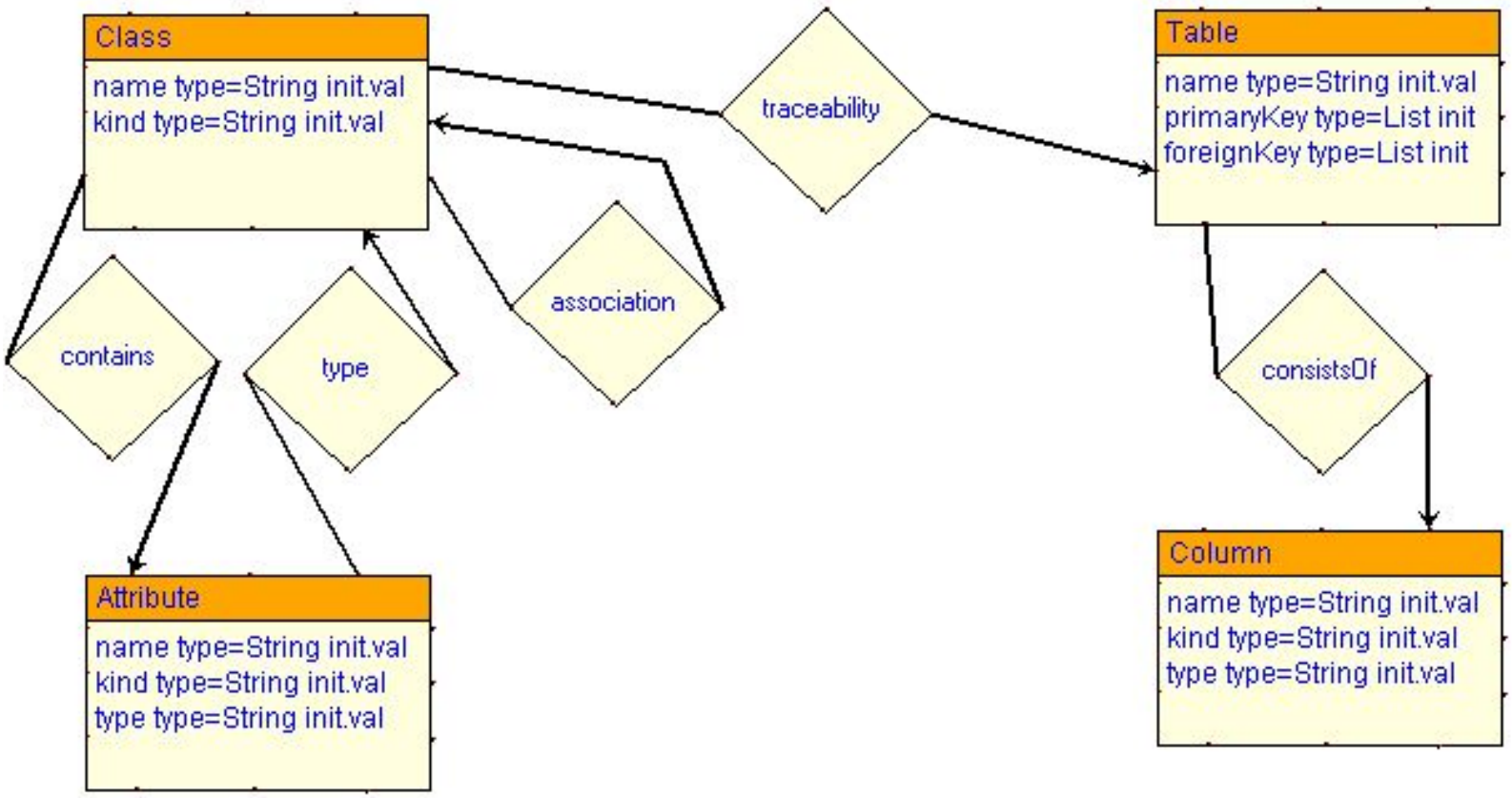
   b) Column kind = “foreign”

   c) Column name = association role + “_” + column name

   d) Attributes of kind = “foreign” are collected in Foreign Key of table

Uml & Rdbms Meta-Model

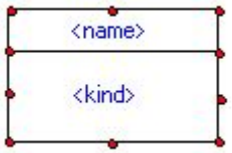
- Common meta-model is described using Entity-Relationship



Uml & Rdbms Meta-Model (II)

- Description of both entity and relationship includes:
  - Attributes
  - Constraints
  - Cardinalities
  - Appearance
- Class cardinalities are:
  - contains dir=Source, 0..N
  - traceability dir=Source, 0..1
  - type dir=Destination, 0..N
  - association dir=Source, 0..N
  - association dir=Destination, 0..N

and appearance:

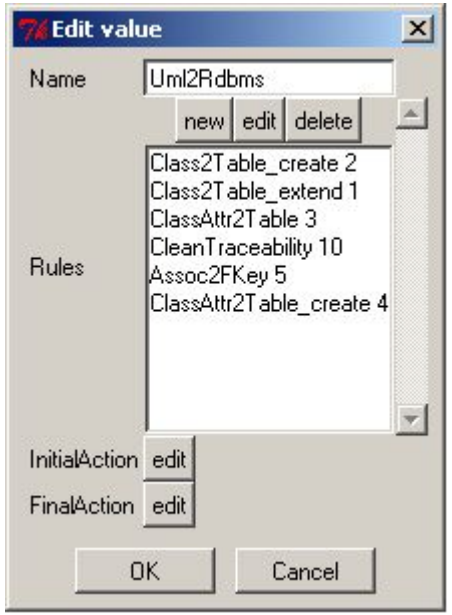


AToM^3 Transformations

- A model transformation in AToM^3 consists of:
  - Initial Action
  - Final Action
  - Set of rules
- Models in AToM^3 are represented as graphs
- Each rule defines how to graph-rewrite left hand side (LHS) to right hand side (RHS)
- LHS is a pattern which is matched against model being transformed
- RHS is a graph that is inserted into the model instead of a matched subgraph
- The complete definition of rule consists of: Name, Order, LHS, RHS, Condition and Action

Uml2Rdbms Transformation

- Uml2Rdbms transformation has six rules:
  - Class2Table_create – creates Table for persistent Class and traceability link between them
  - Class2Table_extend – adds columns for primitive type attributes
  - ClassAttr2Table_create – adds traceability link between the class of the attribute and table
  - ClassAttr2Table – recursively adds columns for attributes of the type Class
  - Assoc2FKey – creates foreign key
  - CleanTraceability – remove all traceability links
- Rules are executed according to their order. Lower order rules can be executed only if none of higher order rules can be applied. If none of the rules can be applied, transformation ends.



Uml2Rdbms Initial Action

- Declaration of lists of processed elements

```
self.rewritingSystem.procClasses = ATOM3List([1, 1, 1, 0], ATOM3String)
self.rewritingSystem.procAttrs = ATOM3List([1, 1, 1, 0], ATOM3String)
self.rewritingSystem.procForeignCols = ATOM3List([1, 1, 1, 0], ATOM3String)
```

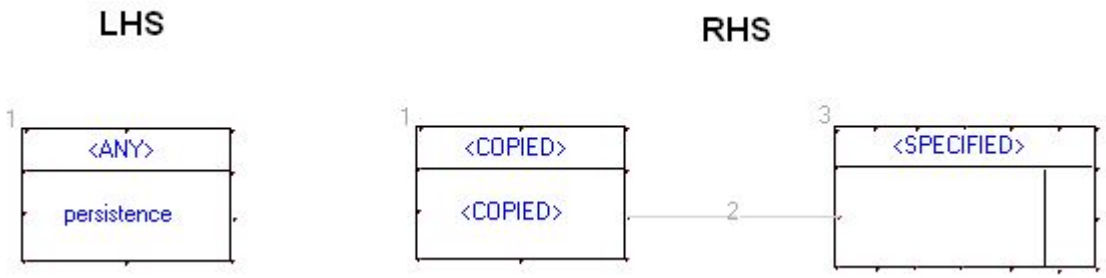
- Processed element's state checkers

```
self.rewritingSystem.isProcClass = lambda cName: [] != filter (
    lambda cn: cn.toString() == cName,
    []+self.rewritingSystem.procClasses.getValue()
)
... isProcAttr and isProcForeignCol that are similar
```

- Add element to processed list

```
self.rewritingSystem.addProcClass = lambda cName:
    self.rewritingSystem.procClasses.setValue(
        [ATOM3String(cName)]+self.rewritingSystem.procClasses.getValue
          ()
    )
... addProcAttr and addProcForeignCol that are similar
```

Class2Table_create Rule



- Condition

```
className = self.getMatched(graphID, self.LHS.nodeWithLabel(1)).
  name.toString()

return not self.graphRewritingSystem.isProcClass(className)
```

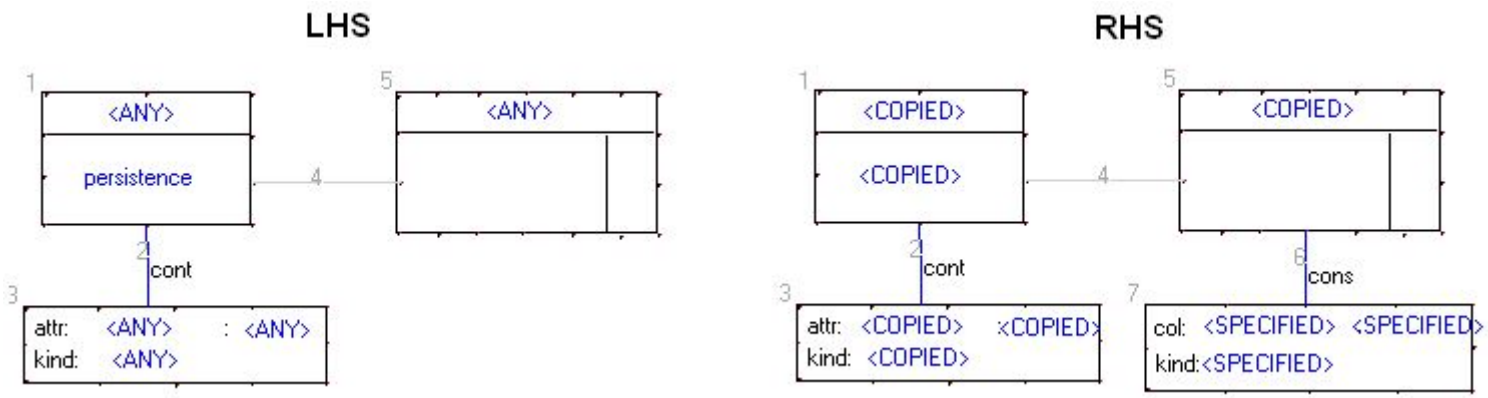
- Action

```
C = self.getMatched(graphID, self.LHS.nodeWithLabel(1)).name.toString
  ()

self.graphRewritingSystem.addProcClass(C)
```

- Table(3).name.AttrSpecify

```
return "t_" + self.getMatched(graphID, self.LHS.nodeWithLabel(1)).
  name.toString()
```

Class2Table_extend Rule



- Column(7).name.AttrSpecify

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(3)).
  name.toString()
```

- Column(7).kind.AttrSpecify

```
return self.getMatched(graphID, self.LHS.nodeWithLabel(3)).
  kind.toString()
```

- Column(7).type.AttrSpecify

```
n = self.getMatched( graphID, self.LHS.nodeWithLabel(3)).
  type.toString()

if n == 'String': return 'VARCHAR'

elif n == 'Integer': return 'NUMBER'

else: return 'unknown'
```

Class2Table_extend Rule (II)

- Condition

```
A = self.getMatched(graphID, self.LHS.nodeWithLabel(3))
if A.type.toString() == 'Class':        return 0
else:
   return not self.graphRewritingSystem.isProcAttr(A.name.toString())
```

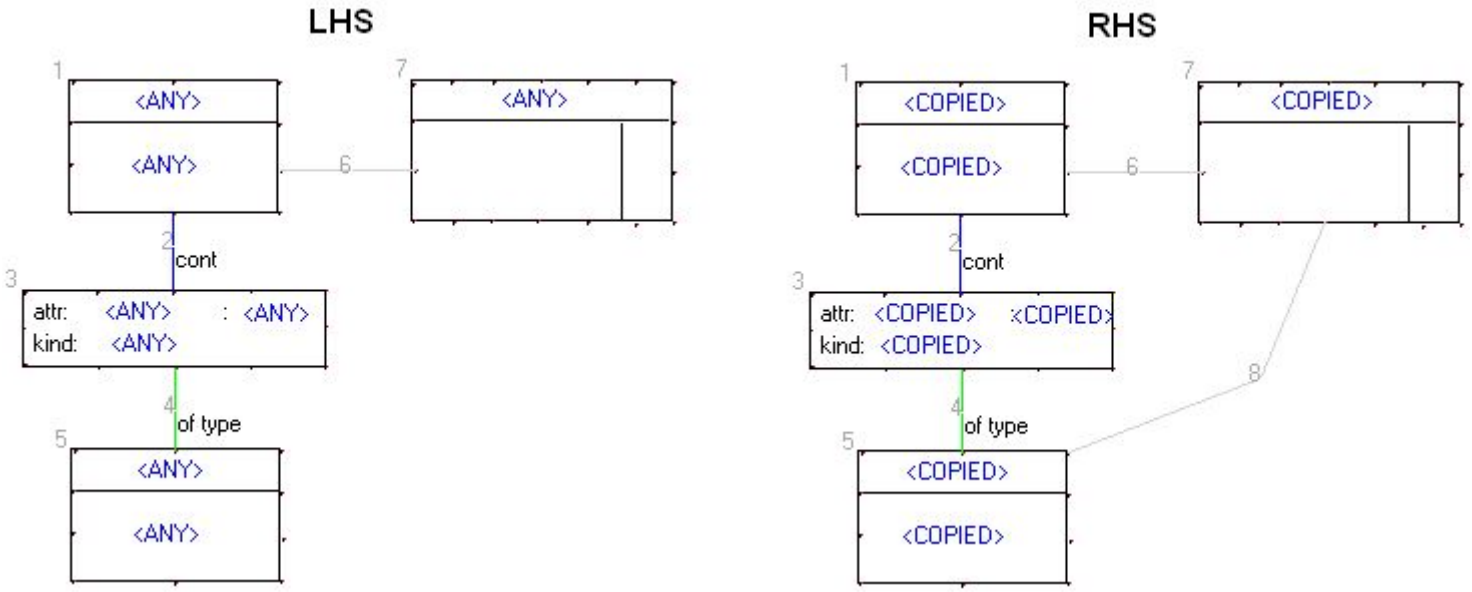
- Table(5).primaryKey.AttrSpecify

```
A = self.getMatched( graphID, self.LHS.nodeWithLabel(3))
T = self.getMatched( graphID, self.LHS.nodeWithLabel(5))
if A.kind.toString() == 'primary':
      return [A.name] + T.primaryKey.getValue()
else: return [] + T.primaryKey.getValue()
```

- Action

```
A = self.getMatched( graphID, self.LHS.nodeWithLabel(3)).
  name.toString()
self.graphRewritingSystem.addProcAttr(A)
```

ClassAttr2Table_create Rule



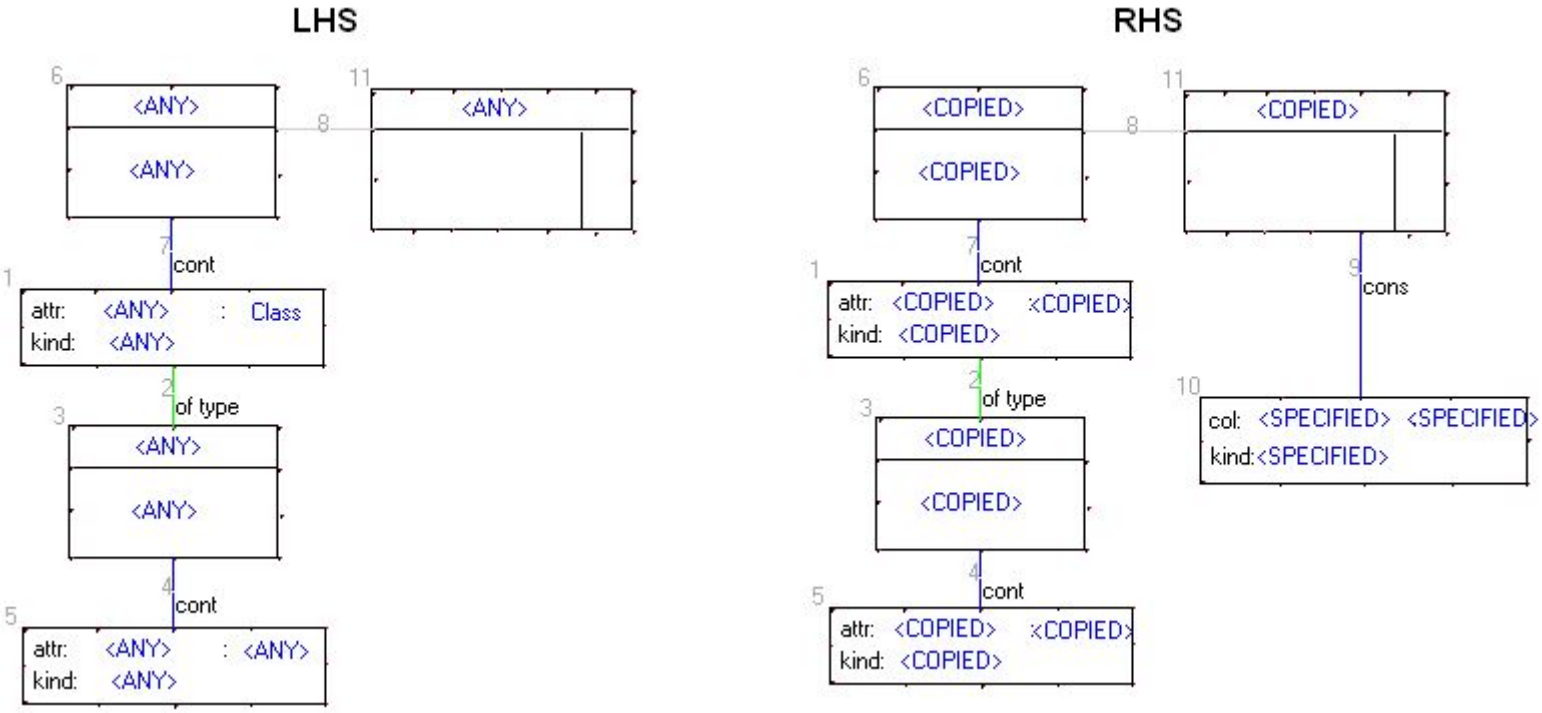
- Condition

```
className = self.getMatched(graphID, self.LHS.nodeWithLabel(5)).
  name.toString()

return not self.graphRewritingSystem.isProcClass(className)
```

- Action

```
className = self.getMatched(graphID, self.LHS.nodeWithLabel(5)).
  name.toString()

self.graphRewritingSystem.addProcClass(className)
```

ClassAttr2Table Rule



- The new column's name, type and kind are set the same way as in Class2Table_extend rule (instead of node 3, node 5 is taken)

ClassAttr2Table Rule (II)

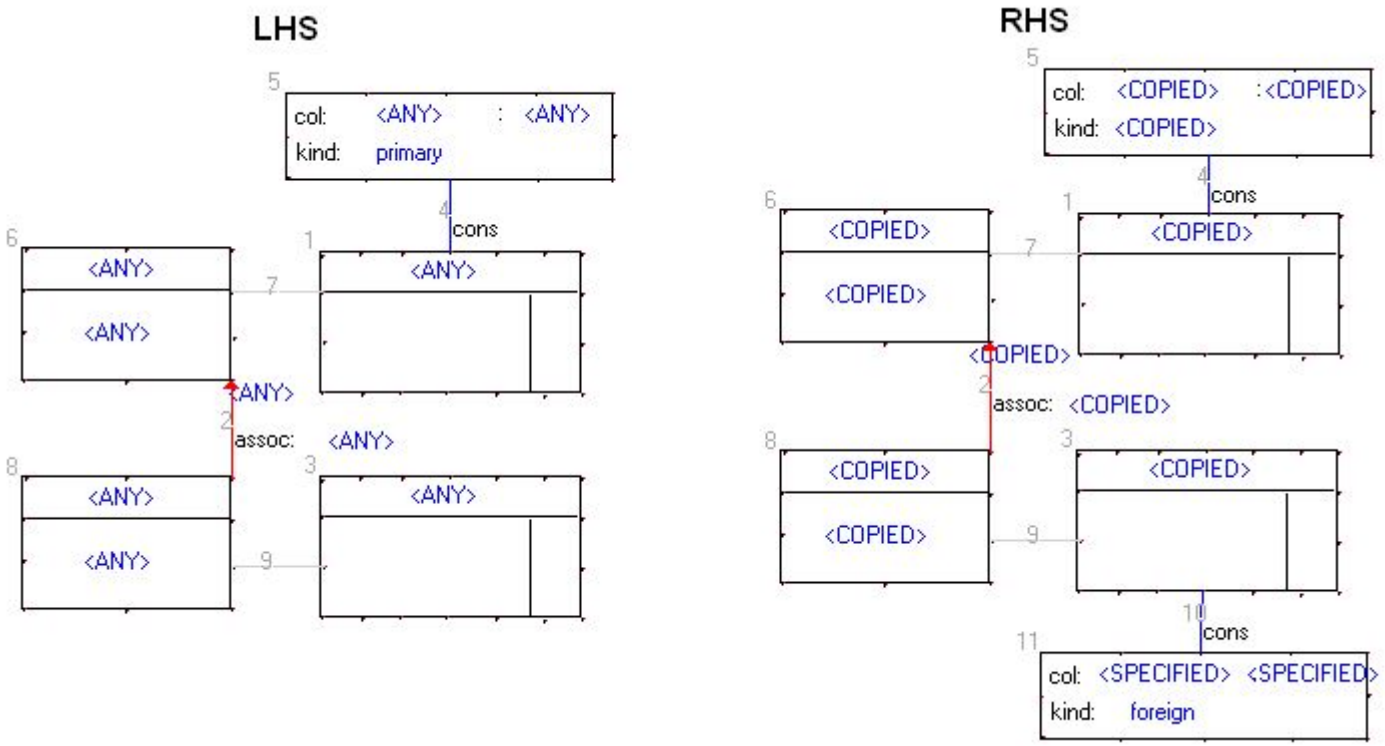
- Condition

```
A = self.getMatched( graphID, self.LHS.nodeWithLabel(5))
if A.type.toString() == 'Class':
  return 0
else:
  return not self.graphRewritingSystem.isProcAttr(A.name.toString())
```

- Action

```
A = self.getMatched( graphID, self.LHS.nodeWithLabel(1))
A2 = self.getMatched( graphID, self.LHS.nodeWithLabel(5))
if A.kind.toString() == 'primary':
   T = self.getMatched( graphID, self.LHS.nodeWithLabel(11))
   newName = A.name.toString() + '_' + A2.name.toString()
   T.primaryKey.setValue(
        [ATOM3String(newName)] + T.primaryKey.getValue() )
self.graphRewritingSystem.addProcAttr(A2.name.toString())
```

Assoc2FKey Rule



- Column(11).name.AttrSpecify

```
AT = self.getMatched( graphID, self.LHS.nodeWithLabel(2)).
   role.toString()

C = self.getMatched( graphID, self.LHS.nodeWithLabel(5)).
   name.toString()

return AT + '_' + C
```

Assoc2FKey Rule (II)

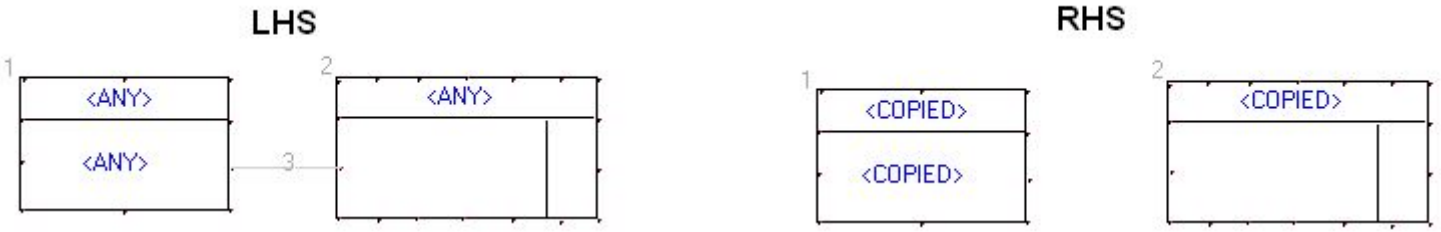
- Condition

```
C = self.getMatched( graphID, self.LHS.nodeWithLabel(5))
if C.kind.toString() != 'primary':
  return 0
else:
  return not self.graphRewritingSystem.isProcForeignCol
   (C.name.toString())
```

- Action

```
CName = self.getMatched( graphID, self.LHS.nodeWithLabel(5)).
  name.toString()
self.graphRewritingSystem.addProcForeignCol(CName)
ATRole = self.getMatched( graphID, self.LHS.nodeWithLabel(2)).
  role.toString()
T = self.getMatched( graphID, self.LHS.nodeWithLabel(3))
newName = ATRole + '_' + CName
T.foreignKey.setValue(
    [ATOM3String(newName)] + T.foreignKey.getValue() )
```

CleanTraceability Rule



- This rule is executed only when no other rule can be applied and it simply removes all traceability links

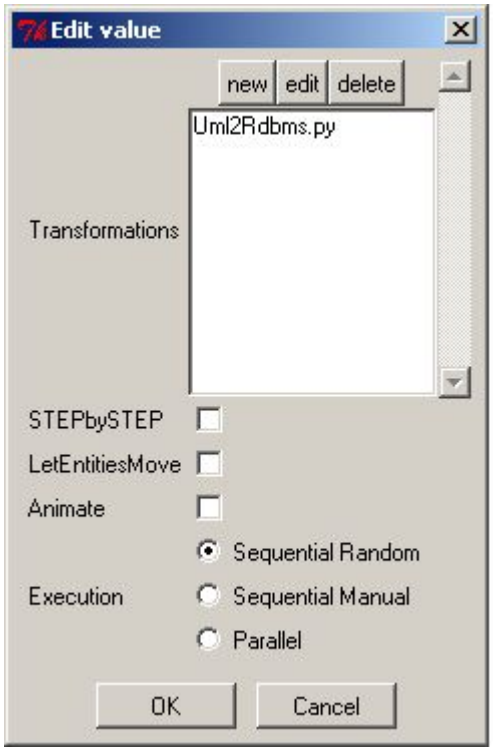
Uml2Rdbms Final Action

- Displays transformation summary

```
print 'Transformation execution finished'
print '---'
print 'processed classes:'
for c in self.rewritingSystem.procClasses.getValue():
   print c.toString()
print '---'
print 'processed attributes:'
for a in self.rewritingSystem.procAttrs.getValue():
   print a.toString()
print '---'
print 'processed foreign key attributes'
for c in self.rewritingSystem.procForeignCols.getValue():
   print c.toString()
```

Running A Transformation

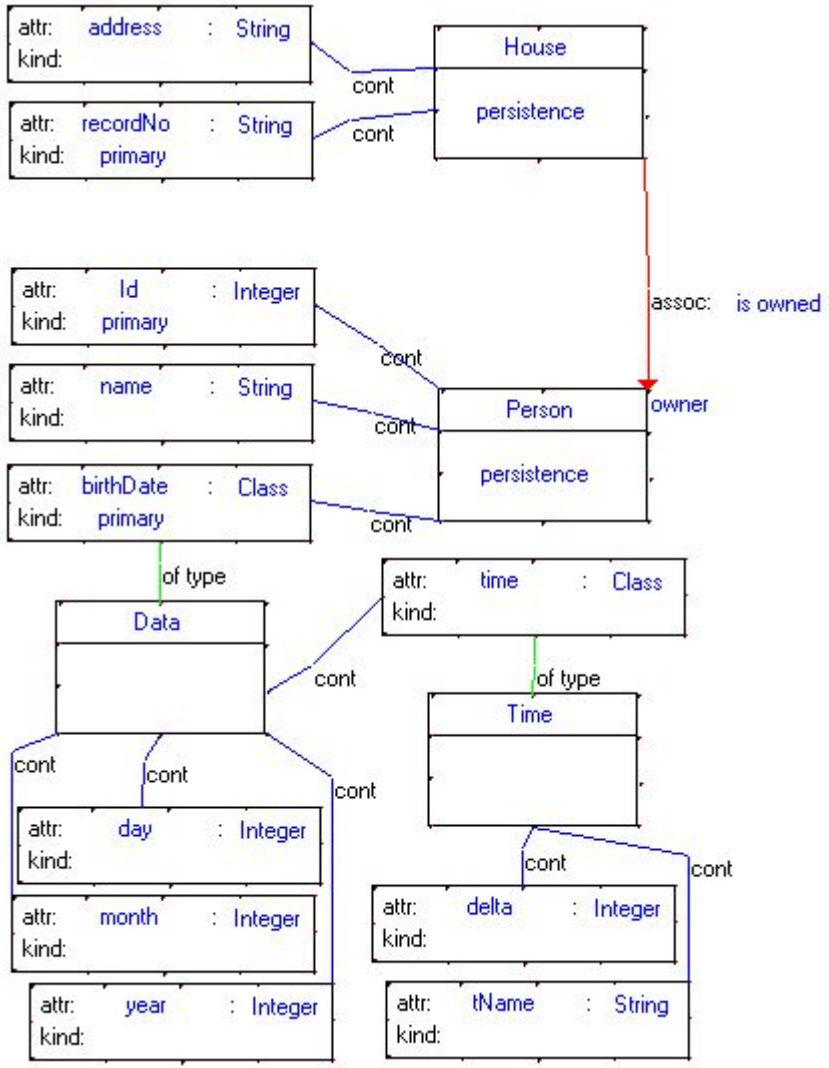
- STEPbySTEP
- Execution modes:
  - Sequential Random – a rule is executed against randomly chosen matched subgraph
  - Sequential Manual – a rule is executed against subgraph chosen by user
  - Parallel – a rule is executed against all separate matched subgraphs in one step



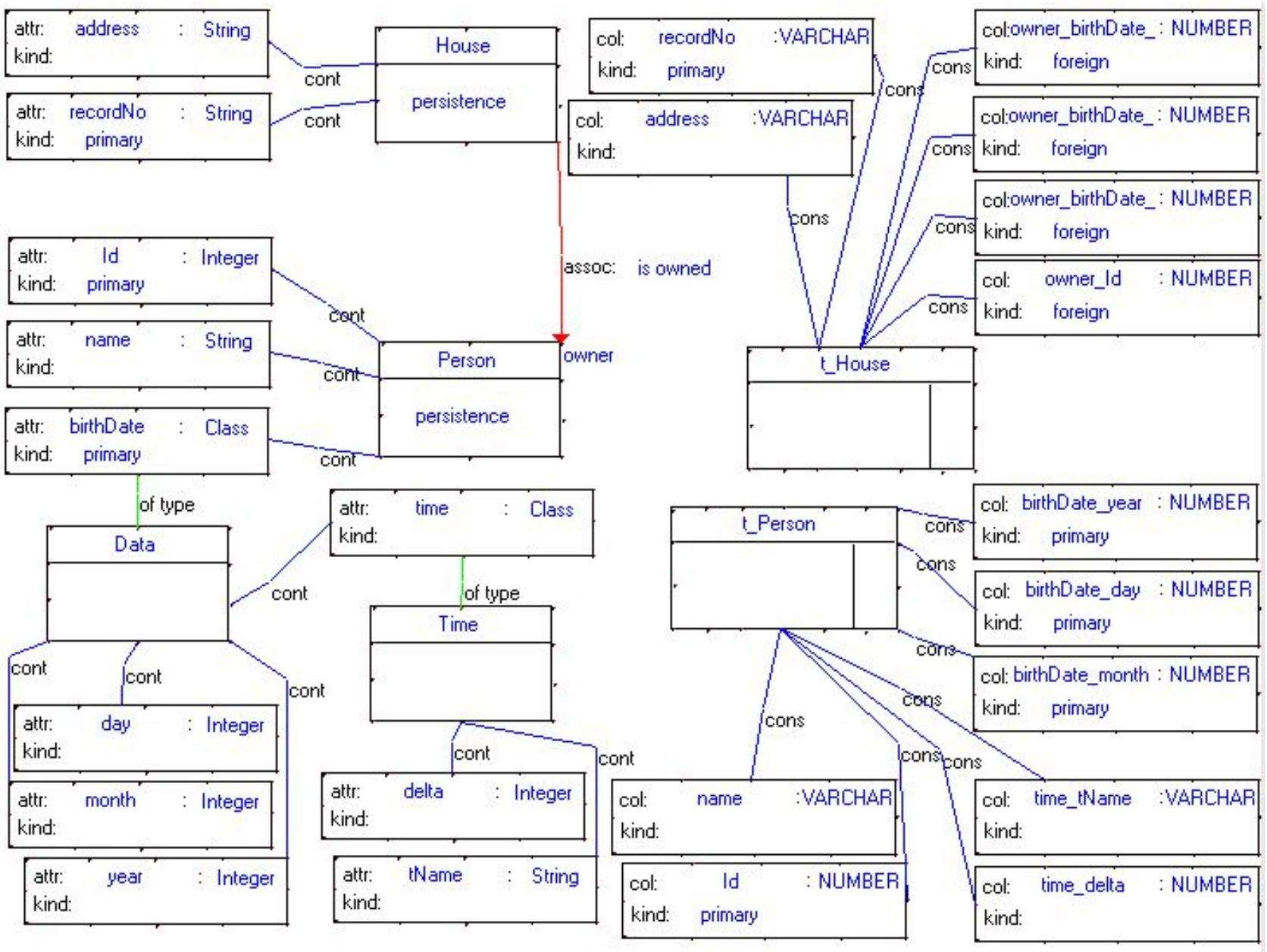
Demo

Expected results:

- t_House.PrimaryKey:
  - recordNo
- t_Person.PrimaryKey:
  - Id
  - birthDate_day
  - birthDate_month
  - birthDate_year
- t_House.ForeignKey:
  - owner_Id
  - owner_birthDate_day
  - owner_birthDate_month
  - owner_birthDate_year

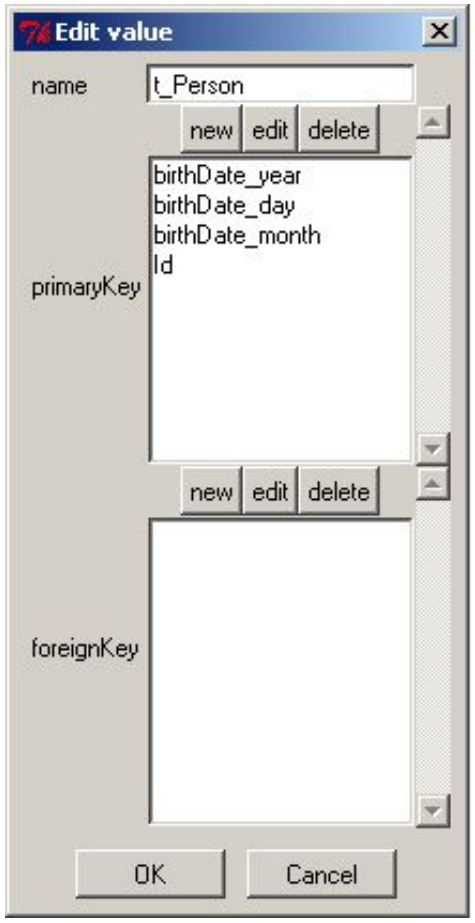


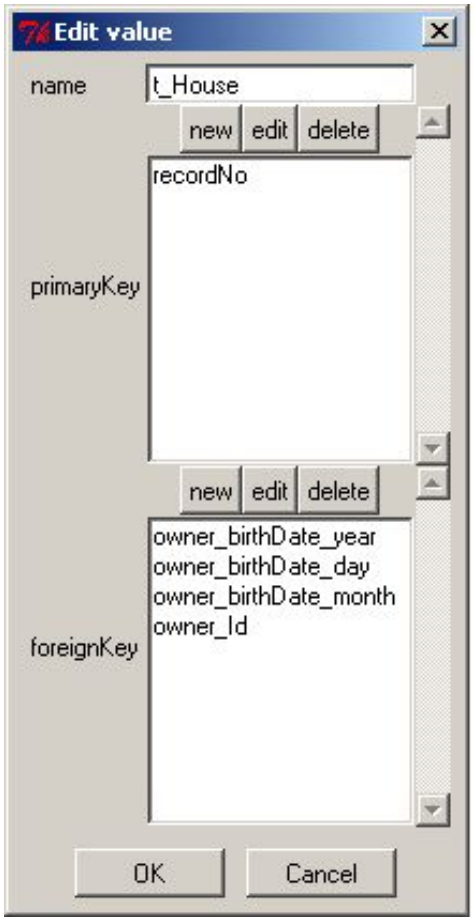
Transformation Result



Transformation Result (II)

- Properties of
  - t_Person table
  - t_House table





Rule Execution Order

- Beginning of execution

ATOM3:> Executing transformation Uml2Rdbms

ATOM3:> Trying rule 1, 2

ATOM3:> Rule 2(Class2Table_create) was executed!

ATOM3:> Trying rule 1

ATOM3:> Rule 1(Class2Table_extend) was executed!

ATOM3:> Trying rule 1

ATOM3:> Rule 1(Class2Table_extend) was executed!

ATOM3:> Trying rule 1, 2

ATOM3:> Rule 2(Class2Table_create) was executed!

ATOM3:> Trying rule 1

ATOM3:> Rule 1(Class2Table_extend) was executed!

ATOM3:> Trying rule 1

ATOM3:> Rule 1(Class2Table_extend) was executed!

ATOM3:> Trying rule 1, 2, 3

ATOM3:> Rule 3(ClassAttr2Table) was executed!

ATOM3:> Trying rule 1, 2, 3

ATOM3:> Rule 3(ClassAttr2Table) was executed!

- End of execution

ATOM3:> Trying rule 1, 2, 3

ATOM3:> Rule 3(ClassAttr2Table) was executed!

ATOM3:> Trying rule 1, 2, 3

ATOM3:> Rule 3(ClassAttr2Table) was executed!

ATOM3:> Trying rule 1, 2, 3, 4

ATOM3:> Rule 4(ClassAttr2Table_create) was executed!

ATOM3:> Trying rule 1, 2, 3, 4, 5

ATOM3:> Rule 5(Assoc2FKey) was executed!

ATOM3:> Trying rule 1, 2, 3, 4, 5

ATOM3:> Rule 5(Assoc2FKey) was executed!

ATOM3:> Trying rule 1, 2, 3, 4, 5

ATOM3:> Rule 5(Assoc2FKey) was executed!

ATOM3:> Trying rule 1, 2, 3, 4, 5

ATOM3:> Rule 5(Assoc2FKey) was executed!

ATOM3:> Trying rule 1, 2, 3, 4, 5, 10

ATOM3:> Rule 10(CleanTraceability) was executed!

Conclusions – Classification

- Transformation Rules
  - Variables – model elements are not contained in variables
  - Patterns – graphs, concrete graphical and semantically typed
  - Logic – non-executable (constrains on attributes); executable imperative (actions) and executable declarative (LHS, RHS graphs)
  - LHS/RHS Syntactic Separation
  - Unidirectional
  - Parametrized
  - No Intermediate Structures
- Rule Application Scoping
  - No Scoping (rules can be applied to the whole model)

Conclusions – Classification (II)

- Source-Target Relationship
  - Target model is the same as source model. Modifications are in-place and update is destructive
- Rule Application Strategy
  - Non-deterministic (both concurrent and one-point (sequential))
  - Interactive
- Rule Scheduling
  - Explicit, Internal (rule order)
  - Rule Selection – Explicit Condition (higher order before lower order rules)
  - Fixed-point rule iteration (until none of the rules can be applied)
  - No Phasing

Conclusions – Classification (III)

- Rule organization
  - No modularity mechanisms
  - No reuse mechanisms
  - Organizational Structure – rules are independent
- Tracing
  - No Dedicated Support
- Directionality
  - Unidirectional

Missing Features

- Multi-formalism models – saving models and transformations in more than one formalism does not work
- Parallel meta-model and transformation development – adding new attribute to entity or relationship causes the transformation to be invalid. Changes to meta-model should be (when possible) incorporated into transformations
- Results of transformation are not in separate model
- Actions and constraints
  - the lack of transformation global name-space, for creation transformation variables and functions
  - the internal graph data structure of AToM3 is not hidden from the user
  - ATOM3String type is not visible in AttrSpecify

Missing Features (II)

- User interface – not suitable for larger projects
- No modularity and explicit sub-transformation execution (sub-transformation Attribute2column could be used by Class2Table_extend and ClassAttr2Table rules)
- Debugging support – no syntactic analysis of Python code, debugging only during transformation run-time, errors going deeply into the internal structure of AToM^3
- Scheduling
  - the lack of "only once" matching for given element
- Pattern matching
  - the lack of "not" matching