Dipartimento di Ingegneria e Scienze
dell'Informazione e Matematica
Università degli Studi dell'Aquila

Towards Analysing Non-Determinism in Bidirectional Transformations

Romina Eramo

Romeo Marinelli, Alfonso Pierantonio, Gianni Rosa

Goal

To support designers in solving non-determinism in their specifications by detecting the rules that give place to alternative solutions, at design time.

Roadmap

Background

Contribution

Realization

Application

Challenges

Conclusion

Bidirectionality

Bidirectional transformations are useful for maintaining the consistency of two (or more) related sources of information

The relevance of bidirectionality in model transformations has been advocated already in 2005 by OMG by including a bidirectional language in QVT

Bidirectionality

In spite of its relevance, bidirectionality has rarely produced anticipated benefits

WHY?

Non-determinism

Probably, among the reasons why bidirectional transformation had limited success there is the ambivalence concerning *non-bijectivity*

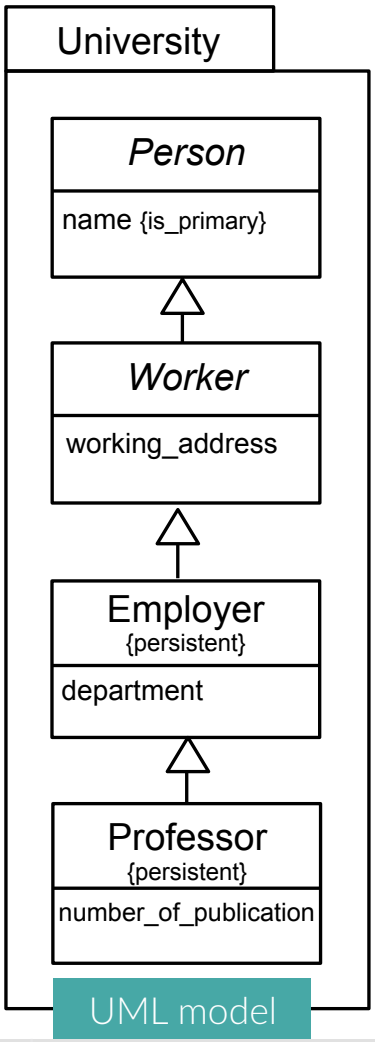
When transformation are *non-bijective*, there may be multiple ways to transform two models into a consistent state

A small example …

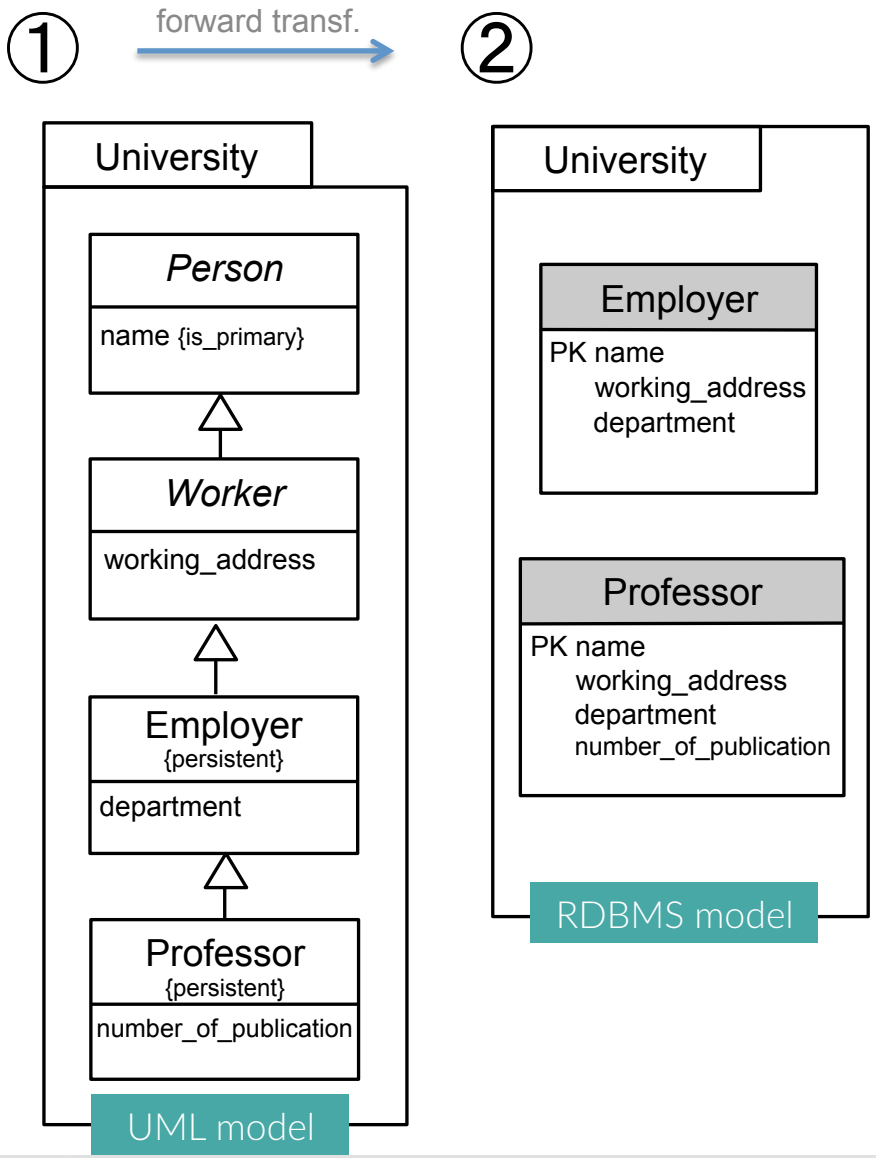
…a typical round-trip problem based on a non-bijective class diagram to relational data base - UML2RDBMS

A small example …

①



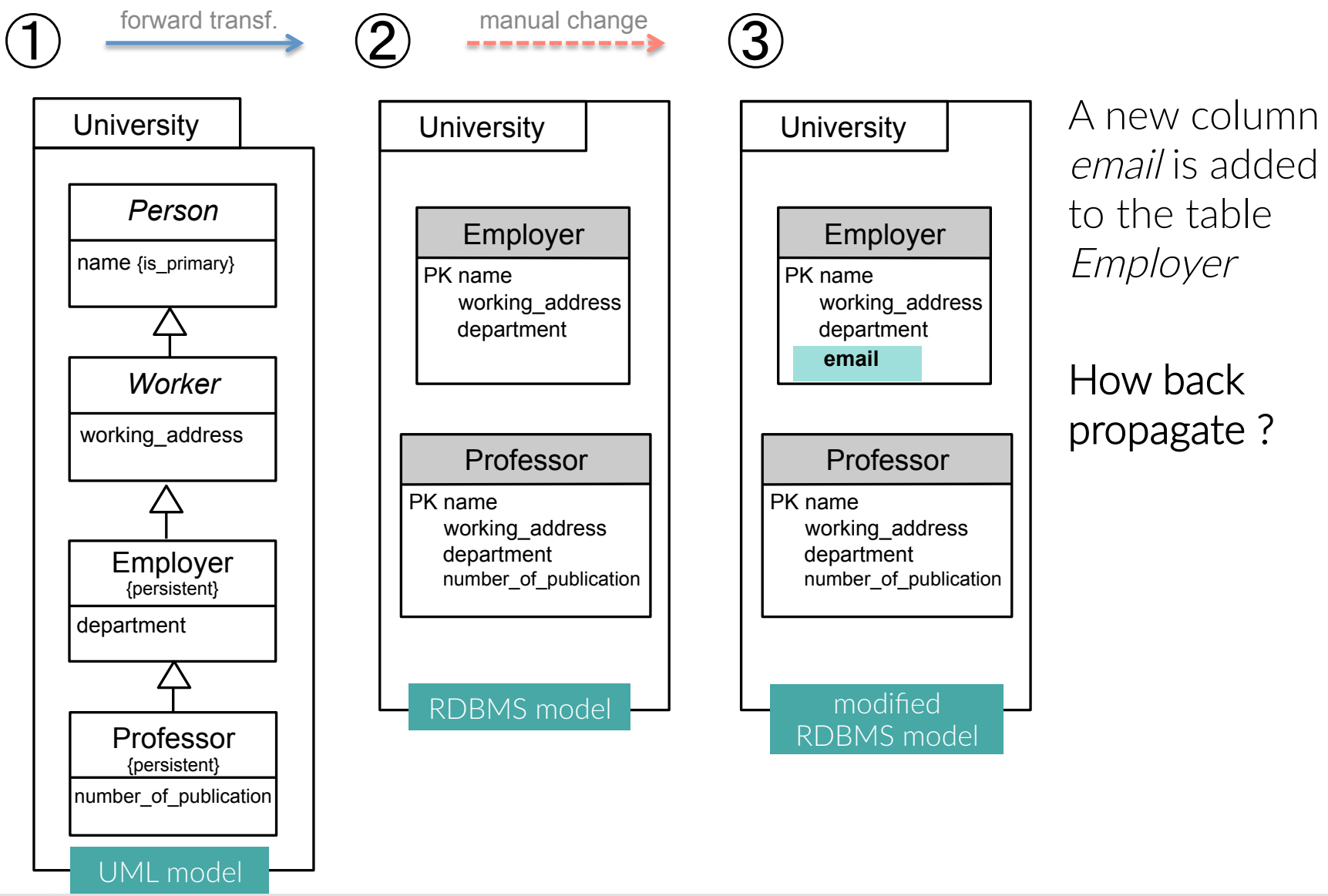
A small example …



Persistent classes are mapped to correspondent tables and attributes to columns, including inherited attributes

Attributes of non-persistent classes are distributed over tables from persistent classes which access non-persistent ones

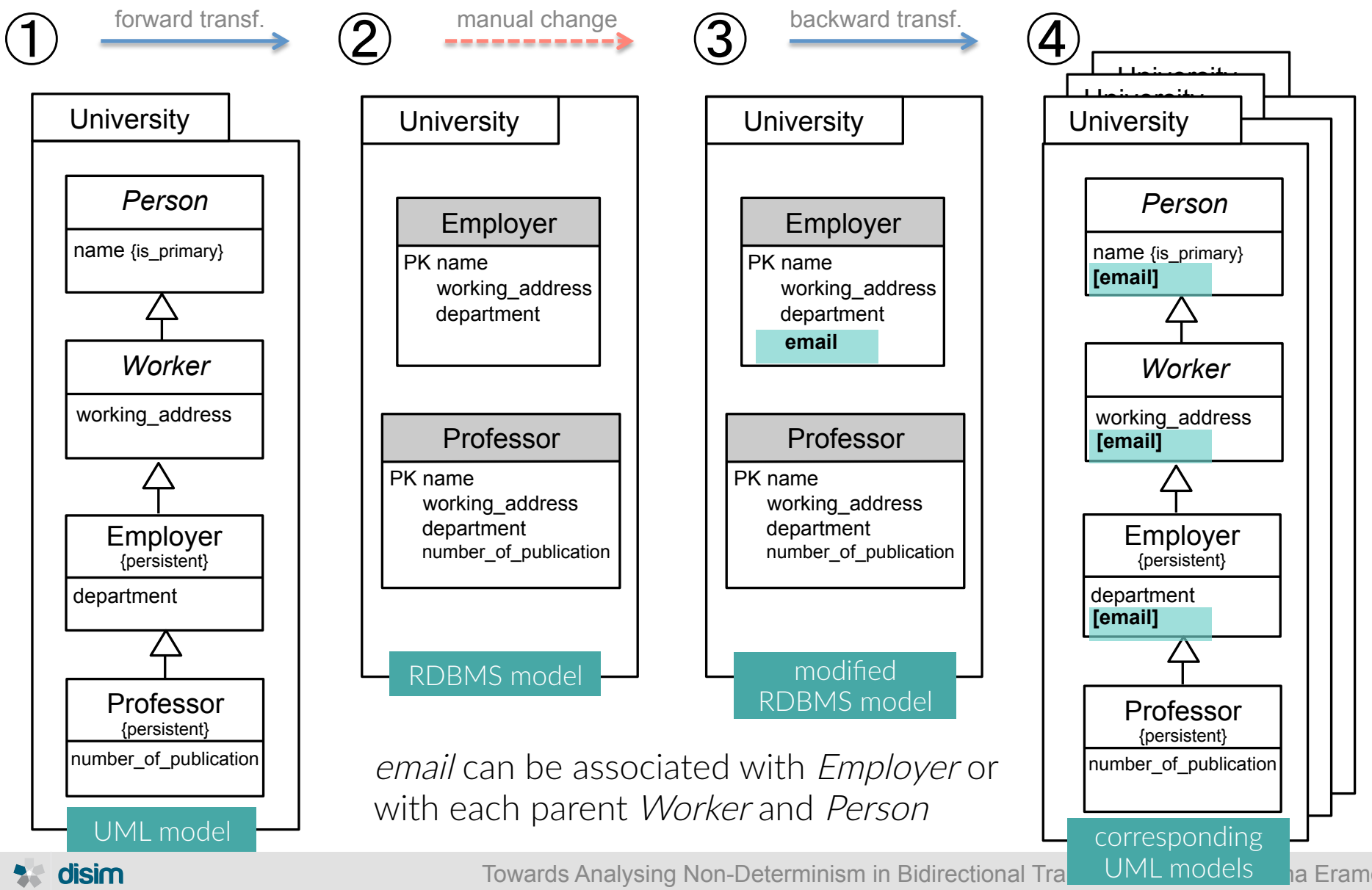
A small example …



A new column *email* is added to the table *Employer*

How back propagate ?

A small example …



email can be associated with *Employer* or with each parent *Worker* and *Person*

Related issues

There have been several works analyzing
semantic issues of bidirectional model transformations

From Stevens 2009 :

*"The developer needs **full control** of what the transformation does. [...] We claim that **determinism** is necessary in order to ensure, first, that developers will find tool behavior predictable, and second, that organisations will not be unacceptably "locked in" to the tool they first use."*

[P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *SOSYM*, 2009]

How it is solved in the current approaches?

1. fix ambiguity via defaults strategy (tool)
2. resolve ambiguity by contraining the specification (designer)
3. generating multiple solutions (tool) and select the desired one (desinger)

Drawbacks:

1. limited control over how models are synchronized
2. early decision and difficult to detect ambiguity
3. onerous management of (large) set of models

Roadmap

Background

Contribution

Realization

Application

Challenges

Conclusion

Contribution

GOAL	To support designers in dealing with non-determinism
HOW	Static analysis of bidirectional transformation
OUTPUT	Degree of non-determinism of the transformation Non-deterministic portions of the transformation
ADVANTAGE	Analysis at design time Detection of code ambiguity before the execution

Roadmap

Background

Contribution

Realization

Application

Challenges

Conclusion

Description of the approach

Analysis notation

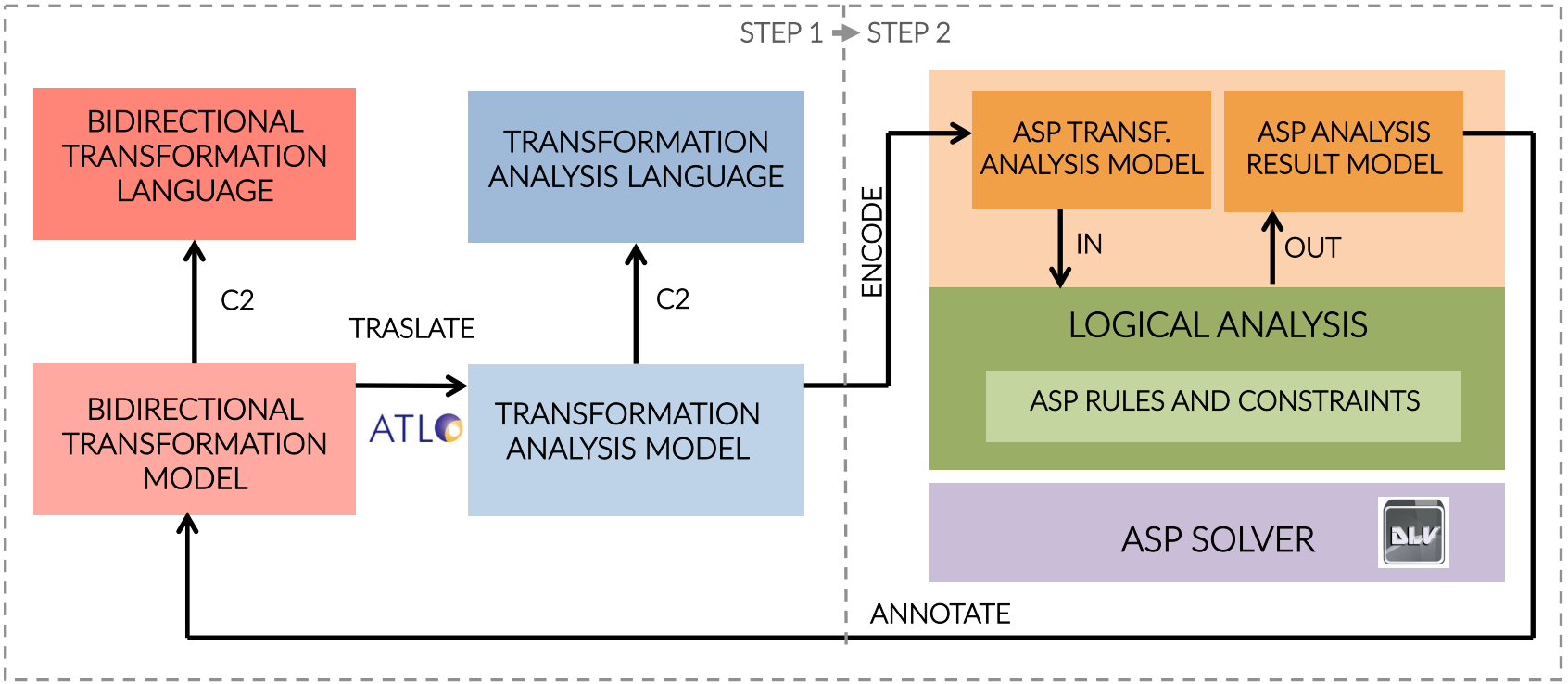
Model transformations are mapped in a format suitable for the analysis. Only relevant information are captured.

Logical rules and constraints

The analysis is performed within a *logic environment* realized in **Answer Set Programming (ASP)**

- the engine is able to analyze the behavior of the transformation with an emphasis on non-determinism

Description of the approach



STEP 1 - Translation from the transformation to the analysis notation

STEP 2 - Execution of the ASP-based analysis and annotation of the transformation

Roadmap

Background

Contribution

Realization

Application

Challenges

Conclusion

Application of the approach

Case study

UML to RDBMS, inheritance hierarchy of classes to corresponding tables, and viceversa

Used transformation language

JTL (Janus Transformation Language), a constraint-based bidirectional language, able to deal with non-bijectivity by generating all the admissible solutions

4 STEPS

1. implementation of the UML2RDBMS bidirectional transf. in JTL
2. transformation of the UML2RDBMS transf. in the analysis language
3. execution of the analysis on the UML2RDBMS analysis model
4. annotation of the UML2RDBMS transf. model with the analysis results

① Implementing UML2RDBMS in JTL

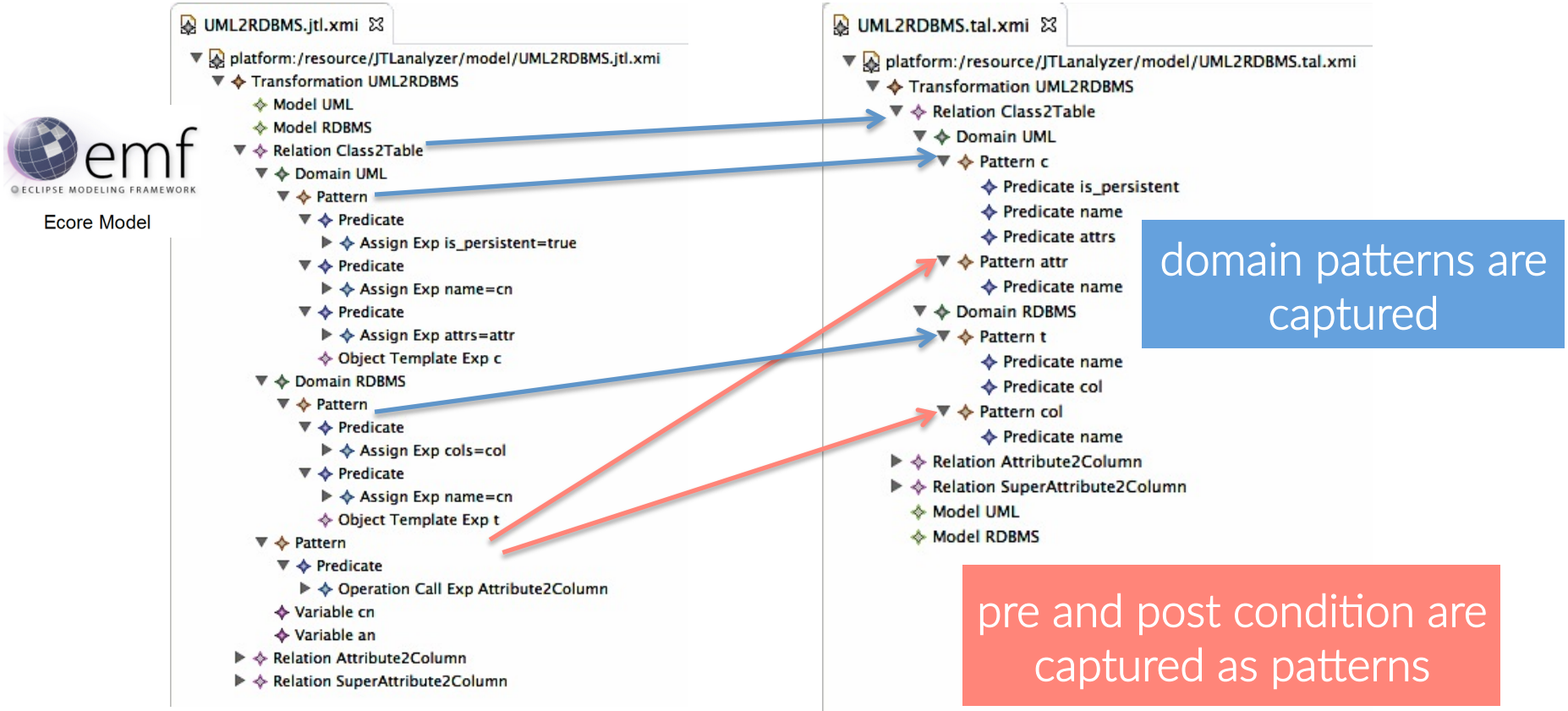
```
1transformation UML2RDBMS(uml:UML, rdbms:RDBMS) { ...
2  top relation Class2Table {
3   cn, an: String;
4   enforce domain uml c:Class {
5     is_persistent = true,
6     name = cn,
7     attrs = attr: Attribute { name = an}
8   };
9   enforce domain rdbms t:Table {
10    name = cn,
11    cols = col: Column { name = an }
12  };
13  when { ... }
14  where { Attribute2Column(c, t); }
15 }
16 relation Attribute2Column {
17  an, at : String;
18  enforce domain uml c:Class {
19    attrs = attr: Attribute { name = an, owner = c, is_primary = false }
20  };
21  enforce domain rdbms t:Table {
22    cols = col: Column { name = an, owner = t }
23  };
24  when { ... }
25 }
26 top relation SuperAttributeToColumn{
27  enforce domain uml c: Class {
28    parent = sc: Class { }
29  };
30  enforce domain rdbms t: Table { };
31  when { ClassToTable(c,t) | (cc = c.parentOf & SuperAttributeToColumn(cc,t));}
32  where { AttributeToColumn(sc, t); }
33 } ...
```

relates classes and tables in the two different metamodels

relates attibutes and columns in the two different metamodels

relates attributes of the parent class to columns of the corresponding child table

② Translating in the analysis notation



A transformation model consists of a set of rules that define mapping among elements types, and pre/post contitions that limit its applicability and behavior

The correspondent transformation analysis model captures only the information which is relevant for performing the analysis, conforming a specific analysis notation

③ Executing the analysis

the analysis model is automatically encoded in ASP

```
1 relation_name(1, class2table).
2 relation_domain(1, uml).
3 relation_pattern(1, uml, c, class).
4 relation_predicate(1, uml, c, is_persistent, true).
5 relation_predicate(1, uml, c, name, cn).
6 relation_predicate(1, uml, c, attrs, attr).   [...]
```

non-determinism is verified by executed ASP rules and constraints on the analysis model used as *base knowledge*

```
 1 are_equal_domains(ID1, ID2, Dom) :-
 2  have_equal_patterns(ID1, ID2, Dom, MC, MCName),
 3  have_equal_predicates(ID1, ID2, Dom, MC, MCName),
 4  not have_different_patterns(ID1, ID2, Dom).
 5
 6 non_deterministic_relation(ID, RelName, DomLeft) :-
 7  are_equal_domains(ID, ID2, DomRight),
 8  relation_name(ID, RelName),
 9  tranformation_model(DomRight),
10  tranformation_model(DomLeft),
11  DomRight != DomLeft.
12
13 are_ambiguous_relations(ID1, ID2, DomLeft) :-
14   are_equal_domains(ID1, ID2, DomRight),
15   tranformation_model(DomRight),
16   tranformation_model(DomLeft),
17   DomRight != DomLef.                          [...]
```

sets of ambiguous transformation rules are detected

```
%%%% sets of non-deterministic relations (RDBMS to UML direction):
[(1,class2table), (3,superAttribute2column), (5,superAttribute2column)]
```

④ Annotating the UML2RDBMS transf.

```
1transformation UML2RDBMS(uml:UML, rdbms:RDBMS) { ...
2  top relation Class2Table {
3   cn, an: String;
4   enforce domain uml c:Class {
5     is_persistent = true,
6     name = cn,
7     attrs = attr: Attribute { name = an}
8   };
9   enforce domain rdbms t:Table {
10    name = cn,
11    cols = col: Column { name = an }
12  };
13  when { ... }
14  where { Attribute2Column(c, t); }
15 }
16 relation Attribute2Column {
17  an, at : String;
18  enforce domain uml c:Class {
19    attrs = attr: Attribute { name = an, owner = c, is_primary = false }
20  };
21  enforce domain rdbms t:Table {
22    cols = col: Column { name = an, owner = t }
23  };
24  when { ... }
25 }
26 top relation SuperAttributeToColumn{
27  enforce domain uml c: Class {
28    parent = sc: Class { }
29  };
30  enforce domain rdbms t: Table { };
31  when { ClassToTable(c,t) | (cc = c.parentOf & SuperAttributeToColumn(cc,t));}
32  where { AttributeToColumn(sc, t); }
33 } ...
```

A set of non-deterministic relations in UML to RDBMS direction is detected

④ Annotating the UML2RDBMS transf.

```
1transformation UML2RDBMS(uml:UML, rdbms:RDBMS) { ...
2  top relation Class2Table {
3   cn, an: String;
4   enforce domain uml c:Class {
5     is_persistent = true,
6     name = cn,
7     attrs = attr: Attribute { name = an}
8   };
9   enforce domain rdbms t:Table {
10    name = cn,
11    cols = col: Column { name = an }
12  };
13  when { ... }
14  where { Attribute2Column(c, t); }
15 }
16 relation Attribute2Column {
17  an, at : String;
18  enforce domain uml c:Class {
19    attrs = attr: Attribute { name = an, owner = c, is_primary = false }
20  };
21  enforce domain rdbms t:Table {
22    cols = col: Column { name = an, owner = t }
23  };
24  when { ... }
25 }
26 top relation SuperAttributeToColumn{
27  enforce domain uml c: Class {
28    parent = sc: Class { }
29  };
30  enforce domain rdbms t: Table { };
31  when { ClassToTable(c,t) | (cc = c.parentOf & SuperAttributeToColumn(cc,t));}
32  where { AttributeToColumn(sc, t); }
33 } ...
```

A set of non-deterministic relations in UML to RDBMS direction is detected

Roadmap

Background

Contribution

Realization

Application

Challenges

Conclusion

Challenges

Language-independent approach

Despite the analysis is completely independent from JTL, in order to make the approach completely language-independent additional implementation efforts are required

Analysis of formal properties

The logical foundation of the engine makes possible the verification of different formal properties by translating them in ASP, with the scope to provide a mean for improving the quality of model transformation specifications

Roadmap

Background

Contribution

Realization

Application

Challenges

Conclusion

Conclusion

Bidirectional transformations are known as a promising approach for maintaining models in a consistent state, their success is limites by semantic issues and intrinsic characteristics

This work represent a first step to support transformation implementors with a tool capable of analyzing transformations in order to return feedback which could resolve potential problem, such as non-determinism

Thanks

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

the model-driven engineering group

Romina Eramo
romina.eramo@univaq.it