A Model Transformation Language based on Logic Programming

Jesús M. Almendros-Jiménez and Luis Iribarne

Universidad de Almería.
{jalmen,liribarne}@ual.es

39th International Conference on Current Trends in Theory and Practice of Computer Science January 26–31, 2013
Spindleruv Mlyn, Czech Republic

Table of contents

Motivation

- Model Driven Software Engineering: MDA, UML, DSLs
- Model Transformation (M2M,M2T), Code Generation

Motivation

- Model Driven Software Engineering: MDA, UML, DSLs
- Model Transformation (M2M,M2T), Code Generation
- Models, Meta-models and Meta-meta-models

Motivation

- Model Driven Software Engineering: MDA, UML, DSLs
- Model Transformation (M2M,M2T), Code Generation
- Models, Meta-models and Meta-meta-models
- Model Transformation Languages: ATL, QVT, ...

Motivation

- Model Driven Software Engineering: MDA, UML, DSLs
- Model Transformation (M2M,M2T), Code Generation
- Models, Meta-models and Meta-meta-models
- Model Transformation Languages: ATL, QVT, ...
- Prolog based languages: VIATRA, Tefkat, MoMaT

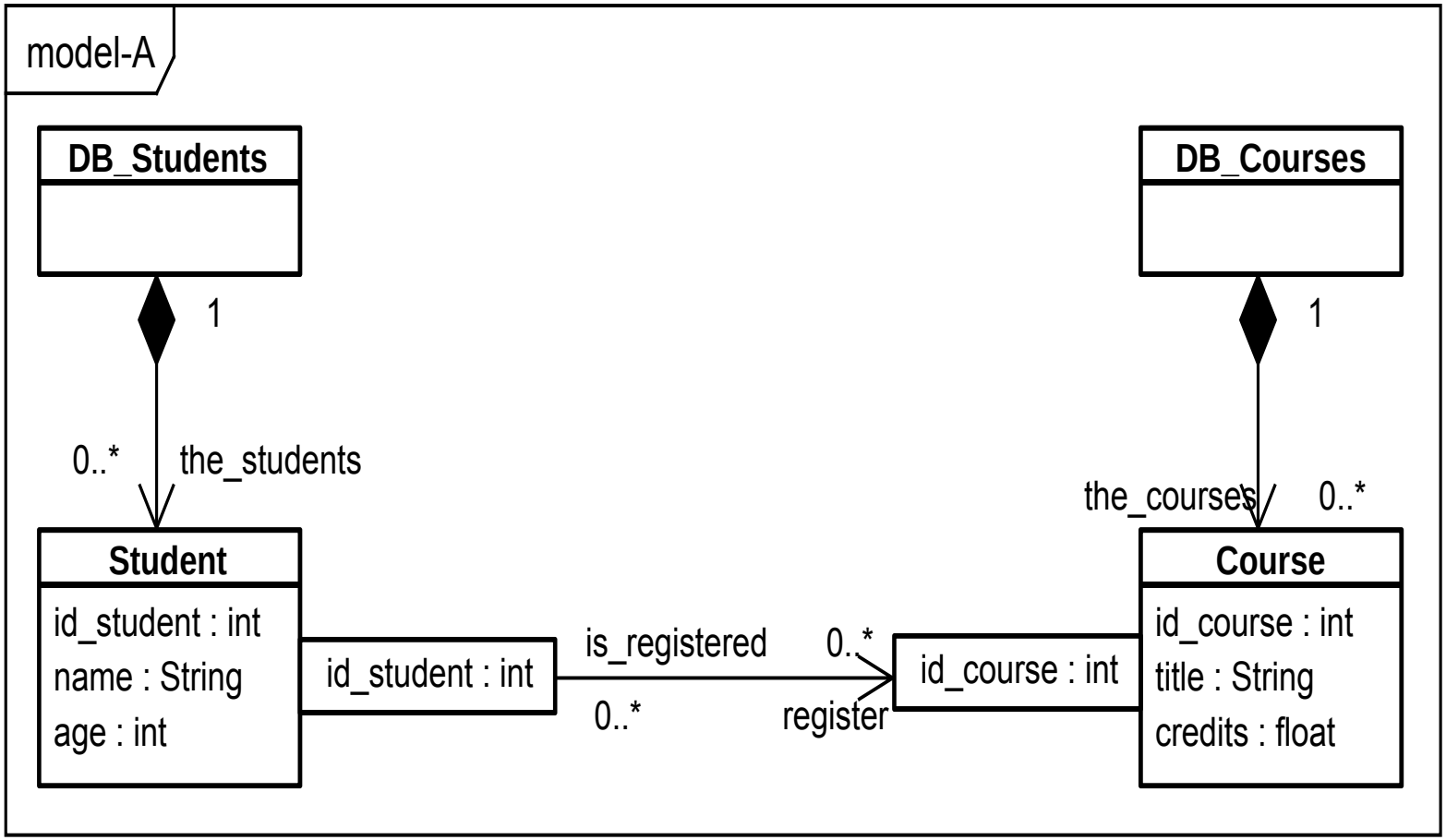
Motivation

- Model Driven Software Engineering: MDA, UML, DSLs
- Model Transformation (M2M,M2T), Code Generation
- Models, Meta-models and Meta-meta-models
- Model Transformation Languages: ATL, QVT, ...
- Prolog based languages: VIATRA, Tefkat, MoMaT
- Prolog as transformation engine for ATL

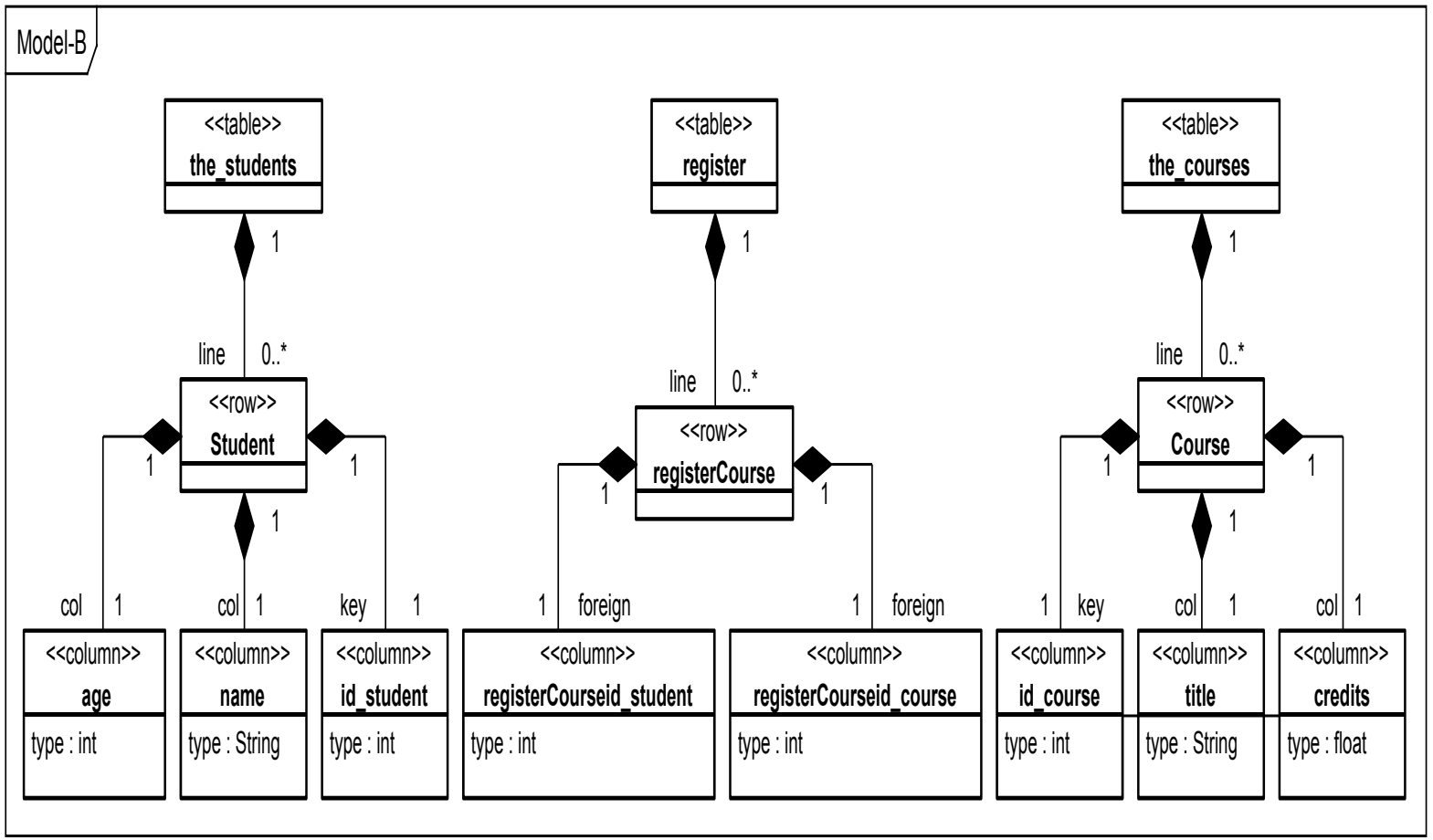
Motivation

- Model Driven Software Engineering: MDA, UML, DSLs
- Model Transformation (M2M,M2T), Code Generation
- Models, Meta-models and Meta-meta-models
- Model Transformation Languages: ATL, QVT, ...
- Prolog based languages: VIATRA, Tefkat, MoMaT
- Prolog as transformation engine for ATL

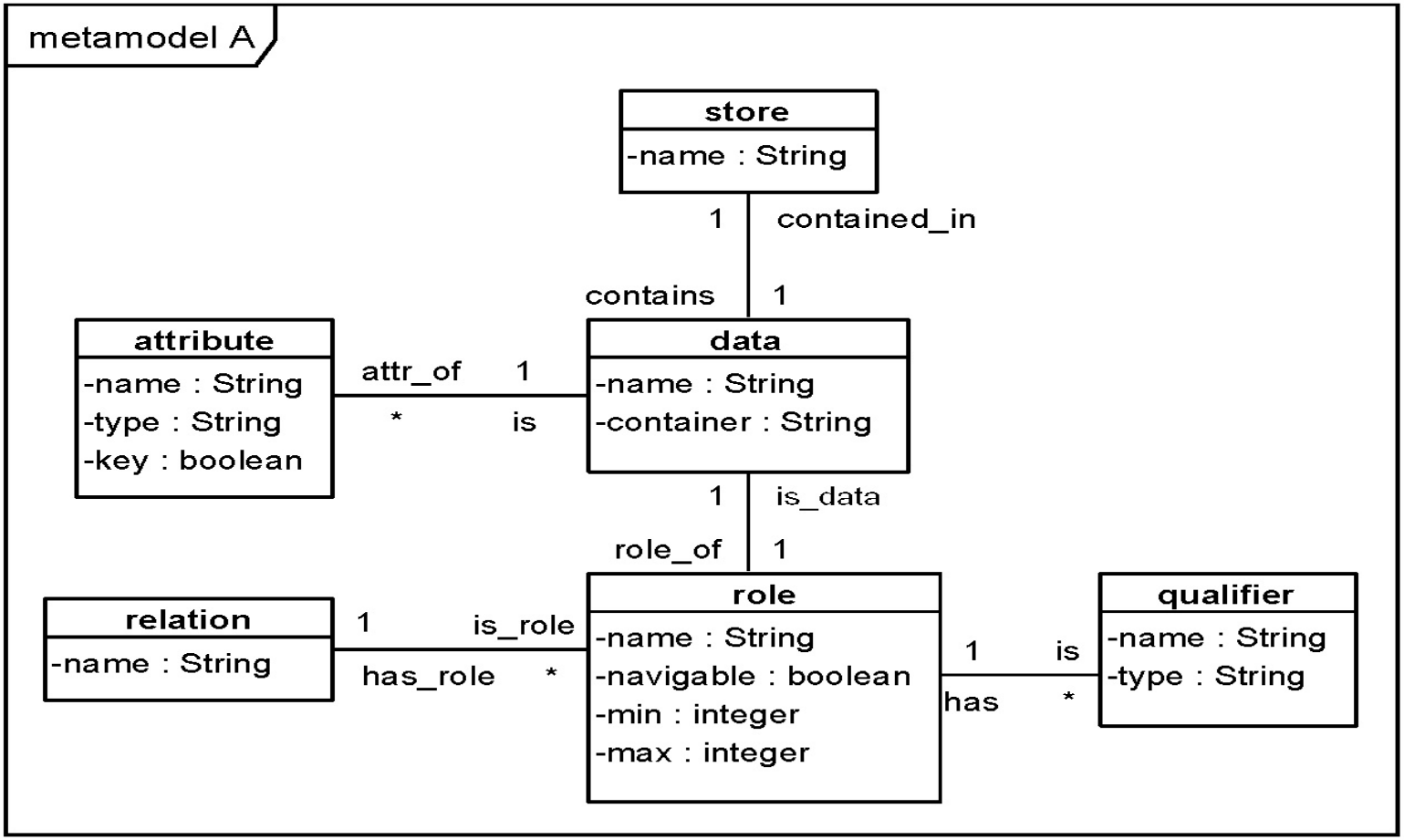
Source Model



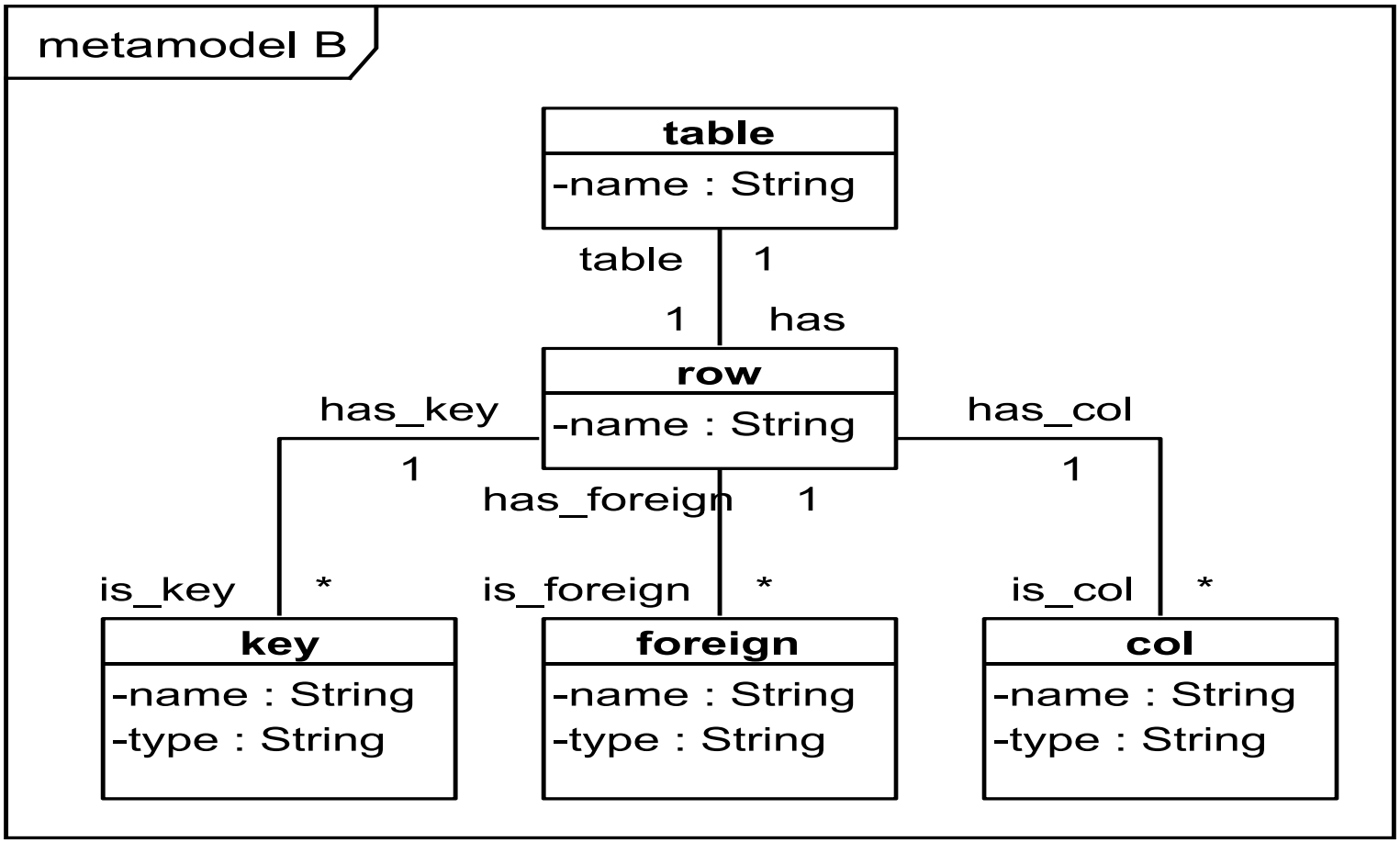
Target Model



Source Meta-Model



Target Meta-Model



PTL features

- ATL style syntax
- Declarative Semantics

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models
- MOF meta-metamodel

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models
- MOF meta-metamodel
- XMI support

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models
- MOF meta-metamodel
- XMI support
- Prolog helpers

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models
- MOF meta-metamodel
- XMI support
- Prolog helpers
- Debugging capabilities

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models
- MOF meta-metamodel
- XMI support
- Prolog helpers
- Debugging capabilities
- Tracing capabilities

PTL features

- ATL style syntax
- Declarative Semantics
- Encoding to Prolog rules
- Prolog library for handling meta-models
- MOF meta-metamodel
- XMI support
- Prolog helpers
- Debugging capabilities
- Tracing capabilities

Meta-model definitions in PTL

```
metamodel(er,
[
        class(data, [name,container]),
        class(store, [name]),
        class(attribute, [name,type,key]),
        class(relation, [name]),
        class(roles, [name,navigable,min,max]),
        class(qualifier, [name,type]),
        role(contains,store,data,"1","1"),
        role(contained_in,data,store,"1","1"),
        role(attr_of,data,attribute,"0","*"),
        role(is,attribute,data,"1","1"),
        role(has_role,roles,relation,"1","1"),
        role(is_role,relation,roles,"1","*"),
        role(has,qualifier,roles,"1","1"),
        role(is,roles,qualifier,"0","*"),
        role(is_data,roles,data,"1","1"),
        role(role_of,data,roles,"1","1")
]
).
```

PTL rule syntax

rule	:= **rule** rule_name **from** *pointers* [**where** *condition*] **to** *objects*
pointers	:= *pointer* | '(' *pointer*,...,*pointer* ')'
condition	:= *bcondition* | *condition* **and** *condition*
bcondition	:= *access* **==** *access* | *access* **=\=** *access*
pointer	:= pointer_name **:** metamodel_name **!** class_name
objects	:= *object*,...,*object*
object	:= *pointer* '(' *binding*,...,*binding* ')'
access	:= value | pointer_name | pointer_name@attribute_name | pointer_name@role_name@attribute_name | **helper_name** '(' *access*,...,*access* ')' | **resolveTemp**'(' '(' *access*,...,*access* ')', pointer_name')' | **sequence**'('[']*access*,...,*access*']''(')'
binding	:= attribute_name <– *access* | role_name <– *access*

PTL rule example

```
rule table2_er2rl from
p:er!role where (p@navigable==true and p@max=="*") to
     (t:rl!table(
        name <- p@name,
        has <- r ),
      r:rl!row(
        name <- concat(p@name,p@is_data@name),
        table <-t,
        is_foreign <- sequence([resolveTemp((p@is,p),f1),
            inverse_qualifier(p)]))
      ).
rule foreign2_er2rl from
(p:er!qualifier,q:er!role) where
        (p@has == q and q@navigable==false) to
        (f2:rl!foreign(
            name <-concat(concat(q@name,q@is_data@name),p@name),
            type <- p@type,
            has_foreign <- inverse_row(p))
         ).
```

PTL helpers

```
inverse_row(A, E) :-
        associationEnds(er, has, A, B),
        associationEnds(er, has_role, B, C),
        associationEnds(er, is_role, C, D),
        role_navigable(er, D, true),
        resolveTemp(D, r, E).
```

PTL Declarative Semantics (I)

Definition (Model)

A model $\mathcal{M}$ is a quintuple $\mathcal{M} = (\mathcal{C}^{\mathcal{M}}, \mathcal{A}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \mathcal{H}^{\mathcal{M}}, \mathcal{O}^{\mathcal{M}})$
(a) $\mathcal{O}^{\mathcal{M}}$ is a set of object names.
(b) $\mathcal{C}^{\mathcal{M}}$ is an interpretation $C_1^{\mathcal{M}}, \ldots, C_n^{\mathcal{M}}$, of the class names which are disjoint subsets of $\mathcal{O}^{\mathcal{M}}$.
(c) $\mathcal{A}^{\mathcal{M}}$ interprets attributes as a set of (partial) functions $att_j^{<\mathcal{M},i>}$ from $\mathcal{O}^{\mathcal{M}}$ to elements of $\mathcal{D}$
(d) $\mathcal{R}^{\mathcal{M}}$ interprets roles as a set of (partial) functions $r_1^{\mathcal{M}}, \ldots, r_m^{\mathcal{M}}$ from $\mathcal{O}^{\mathcal{M}}$ to a subset of $\mathcal{O}^{\mathcal{M}}$.
(e) $\mathcal{H}^{\mathcal{M}}$ interprets helpers as (partial) functions $h_w^{\mathcal{M}} : C_{w_1}^{\mathcal{M}} \ldots C_{w_t}^{\mathcal{M}} \rightarrow C_{w_{(t+1)}}^{\mathcal{M}}$ $w_i \in \{1, \ldots, n\}$.

PTL Declarative Semantics (II)

(1) $[\![$ **rule** rn **from** *ps* **where** *bc* **to** *obs* $]\!]^{\mathcal{M}} =$
$\cup_{\{\overline{v \in C^{\mathcal{M}}}, \overline{(pn, C^{\mathcal{M}})} = <<ps>>^{\mathcal{M}}, [\![bc]\!]^{\mathcal{M}}_{(pn,v)} \text{ is true}\}} [\![obs]\!]^{\mathcal{M}}_{(pn,v)}$

(2) $<< pn : mm!C >>^{\mathcal{M}} = (pn, C^{\mathcal{M}})$ whenever $C^{\mathcal{M}} \in \mathcal{C}$

(3) $<< (ps_1, ..., ps_n) >>^{\mathcal{M}} = \overline{<< ps >>^{\mathcal{M}}}$

(4) $[\![expr_1$ **and** $expr_2]\!]^{\mathcal{M}}_{(pn,v)}$ if $[\![expr_1]\!]^{\mathcal{M}}_{(pn,v)}$ and $[\![expr_2]\!]^{\mathcal{M}}_{(pn,v)}$ then true else false

(5) $[\![expr_1 == expr_2]\!]^{\mathcal{M}}_{(pn,v)}$ if $[\![expr_1]\!]^{\mathcal{M}}_{(pn,v)} = [\![expr_2]\!]^{\mathcal{M}}_{(pn,v)}$ then true else false

(6) $[\![expr_1 = \backslash = expr_2]\!]^{\mathcal{M}}_{(pn,v)}$ if $[\![expr_1]\!]^{\mathcal{M}}_{(p,v)} = [\![expr_2]\!]^{\mathcal{M}}_{(pn,v)}$ then false else true

(7) $[\![qn{:}mm!C(bd_1, ..., bd_n)]\!]^{\mathcal{M}}_{(pn,v)} = \mathcal{M}' \cup_{1 \leq i \leq n} [\![bd_i]\!]^{<\mathcal{M},o>}_{(pn,v)}$
where $\mathcal{M}' = (\{C^{\mathcal{M}'}\}, \{\}, \{\}, \{\}, \{o\})$, $C^{\mathcal{M}'} = \{o\}$ and
$o = gen\_id(\overline{v}, qn)$, whenever $C^{\mathcal{M}} \in \mathcal{C}$

(8) $[\![o_1, ..., o_n]\!]^{\mathcal{M}}_{(pn,v)} = \cup_{i=1,...,n} [\![o_i]\!]^{\mathcal{M}}_{(pn,v)}$

PTL Declarative Semantics (III)

(9) $[\![v]\!]^{\mathcal{M}}_{(pn,v)} = v$

(10) $[\![pn_i]\!]^{\mathcal{M}}_{(pn,v)} = v_i$

(11) $[\![pn_i@att]\!]^{\mathcal{M}}_{(pn,v)} = att^{\mathcal{M}}(v_i)$

(12) $[\![pn_i@r@att]\!]^{\mathcal{M}}_{(pn,v)} = att^{\mathcal{M}}(r^{\mathcal{M}}(v_i))$

(13) $[\![h(expr_1, ..., expr_n)]\!]^{\mathcal{M}}_{(pn,v)} = h^{\mathcal{M}}([\![expr_1]\!]^{\mathcal{M}}_{(pn,v)}, \ldots, [\![expr_n]\!]^{\mathcal{M}}_{(pn,v)})$

(14) $[\![\textbf{resolveTemp}((expr_1, ..., expr_n), qn)]\!]^{\mathcal{M}}_{(pn,v)} = o$
where $([\![expr_1]\!]^{\mathcal{M}}_{(pn,v)}, \ldots, [\![expr_n]\!]^{\mathcal{M}}_{(pn,v)}) \rightarrow^{\mathcal{P}}_{qn} o$

(15) $[\![\textbf{sequence}([\overline{expr}])]\!]^{\mathcal{M}}_{(pn,v)} = \overline{[\![expr]\!]^{\mathcal{M}}_{(pn,v)}}$

(16) $[\![att{<}{-}expr]\!]^{<\mathcal{M},o>}_{(pn,v)} = \mathcal{M}'$,
where $\mathcal{M}' = (\{\}, \{att^{\mathcal{M}'}\}, \{\}, \{\}, \{\})$, $att^{\mathcal{M}'}(o) = [\![expr]\!]^{\mathcal{M}}_{(pn,v)}$

(17) $[\![r{<}{-}expr]\!]^{<\mathcal{M},o>}_{(pn,v)} = \mathcal{M}'$,
where $\mathcal{M}' = (\{\}, \{\}, \{r^{\mathcal{M}'}\}, \{\}, \{\})$, $r^{\mathcal{M}'}(o) = [\![expr]\!]^{\mathcal{M}}_{(pn,v)}$

PTL Declarative Semantics (IV)

Theorem (Soundness and Completeness)

Given a PTL program $\mathcal{P}$, a model $\mathcal{M}$ that interprets the meta-models of $\mathcal{P}$, and $\mathcal{M}' = \cup_{1 \leq i \leq n} [\![r_i]\!]^{\mathcal{M}}$, where $\mathcal{R}^{\mathcal{P}} \equiv r_1, \ldots, r_n$, then:
$o \in \mathcal{O}^{\mathcal{M}'}$ iff o is a logic consequence of the program $enc(\mathcal{R}^{\mathcal{P}}) \cup \mathcal{H}^{\mathcal{P}} \cup enc(\mathcal{M}) \cup \mathcal{MM}^{\mathcal{P}}$.

Prolog library for meta-models

```
role_id(er, A) :-
      role(er, A, [name(_), navigable(_), min(_), max(_)]).
data_container(er, A, B) :-
      data(er, A, [name(_), container(B)]).
attribute_type(er, A, B) :-
      attribute(er, A, [name(_), type(B), key(_)]).
qualifier_has(er, A, B) :-
      associationEnds(er, has, A, B).
relation_is_role(er, A, B) :-
      associationEnds(er, is_role, A, B).
```

Encoding to Prolog

```
object(rl, foreign, L, (A, B), [name(K), type(C)], f2) :-
        foreign2_er2rl((A, B)),
        qualifier_type(er, A, C),
        qualifier_name(er, A, I),
        role_is_data(er, B, E),
        data_name(er, E, G),
        role_name(er, B, F),
        concat(F, G, H),
        concat(H, I, K),
        generate_ids((A, B), f2, L).

foreign2_er2rl((A, B)) :-
        qualifier_id(er, A),
        role_id(er, B),
        qualifier_has(er, A, B),
        role_navigable(er, B, false).
```

PTL interpreter

```
ptl(Program):-[Program],
              generate_metamodels,
              generate_rules,
              load_models,
              clean_transformation.
```

PTL interpreter

```
load_model(A) :-object(A, B, C, D, E, F),
        assert(objectM(A, B, C, D, E, F)),
        fail.
load_model(A) :-associationObjects(A, B, C, D),
        assert(associationObjectsM(A, B, C, D)),
        fail.
load_model(_).
```

Running a transformation

```
?- transform(['mm2er.ptl','er2rl.ptl','rl2mm.ptl']).
```

Running a transformation

```
transform(Files):-clean_ptl,
                  transform_files(Files).
transform_files([]):-!.
transform_files([F|RF]):-ptl(F),
                  transform_files(RF).
```

Debugger

```
?- debugging(['mm2er.ptl','er2rl.ptl','rl2mm.ptl']).
Debugger: Rule Condition of: table2_er2rl cannot be satisfied.
Found error in: role_name
```

Debugger

```
?-  debugging(['mm2er.ptl','er2rl.ptl','rl2mm.ptl']).
Debugger: Objects of: table2_er2rl cannot be created.
Found error in: resolveTemp
```

Tracer

```
?- tracing(['mm2er.ptl','er2rl.ptl','rl2mm.ptl'],'275284307
    q275325284r2f2c').
Tracing the element: 275284307q275325284r2f2c

Rule: foreign1_rl2mm
Element: foreign
Metamodel: rl

Rule: foreign2_er2rl
Element: qualifier
Metamodel: er
....
Element: property
Metamodel: mm
xmi:id is wfP6O_iGCkzsbgVr

Element: property
Metamodel: mm
xmi:id is CrZGO_iGCkzsbhYY
```

Conclusions and Future Work

- PTL: Prolog based Transformation Language
- Details can be found at `http://indalog.ual.es/mdd`
- Declarative Semantics, Debugging, Tracing.
- Future: Model validation, rule validation, etc
- Future: More about debugging and tracing
- Future: Eclipse plugin for transformation