Designing relational schema from an object model using model transformations

Antoine Wiedemann[1]

[1]Laboratoire informatique image et interaction, Université de La Rochelle

International Conference on Model Driven Engineering Languages and Systems, Doctoral Symposium

Outline

Relational Schema Design

Automatic generation of a relational schema from a domain model.

Key features

- Fully UML compliant
- User inputs are assisted, reduced to the minimum
- We aim at a non redundant relational schema, near to a normal form (BCNF)

No other approaches has the same set of features.

- E/R diagrams
  - separate notation
  - no NULL values ([0..1] multiplicity)
- Design propositions[1]: no automation
- ORM tools ...

[1] Paul Dorsey, Martin Fowler, Susan D. Urban

FAQ : How close are you from Hibernate and al...?

- We do not provide a "object/relational persistence and query service" for Java objects.
- We provide a relational schema design tool, based on domain object model
- We do take care of the final schema quality, i.e. non-redundancy and consistency with the domain model
- The output schema may be used by several applications (object-based or not)

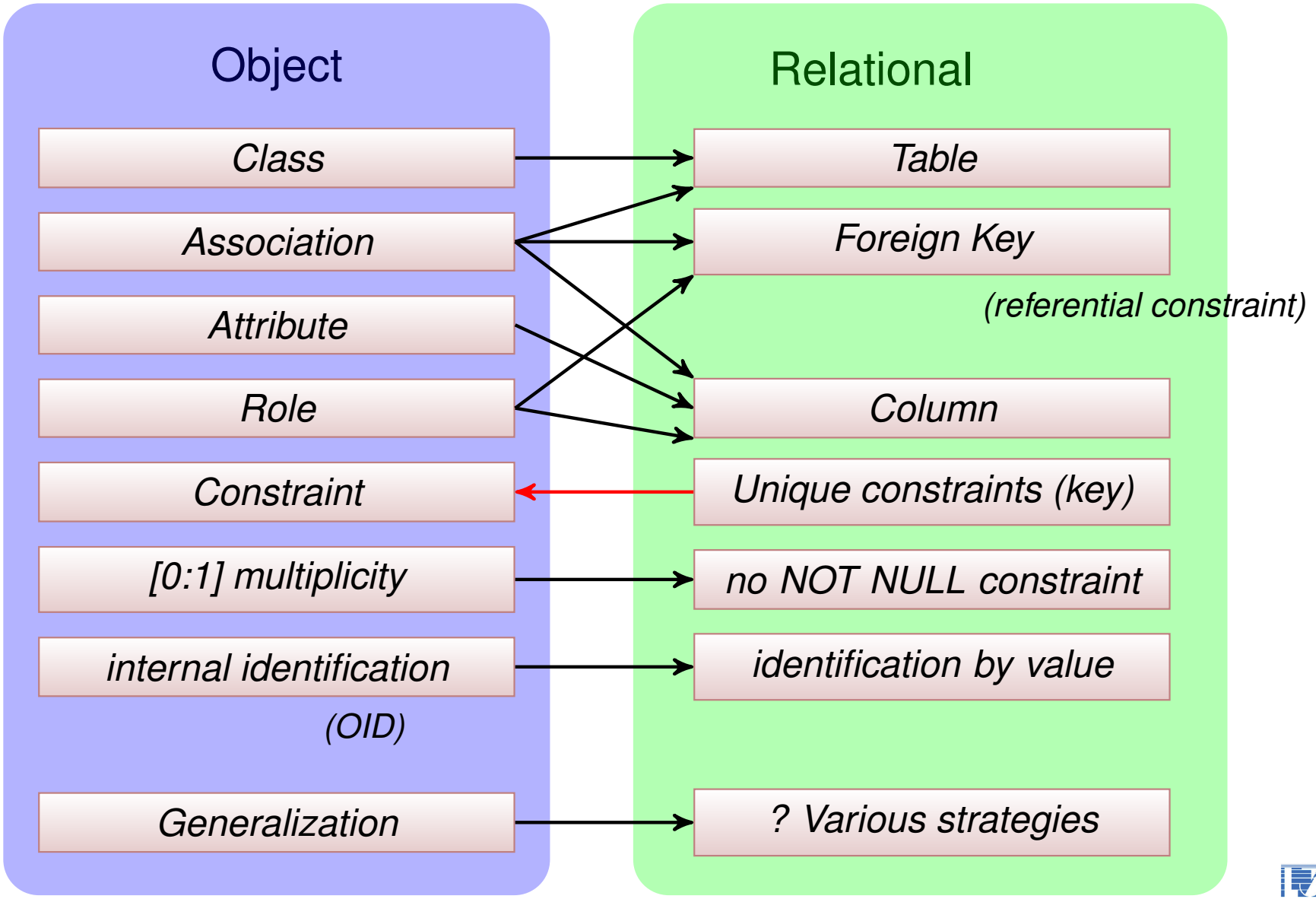
Hibernate workflow

1. Program classes
2. Map classes to persistence layer
3. Persist objects

Our workflow

1. Model the domain
2. Add constraints to models
3. Produce a relational schema

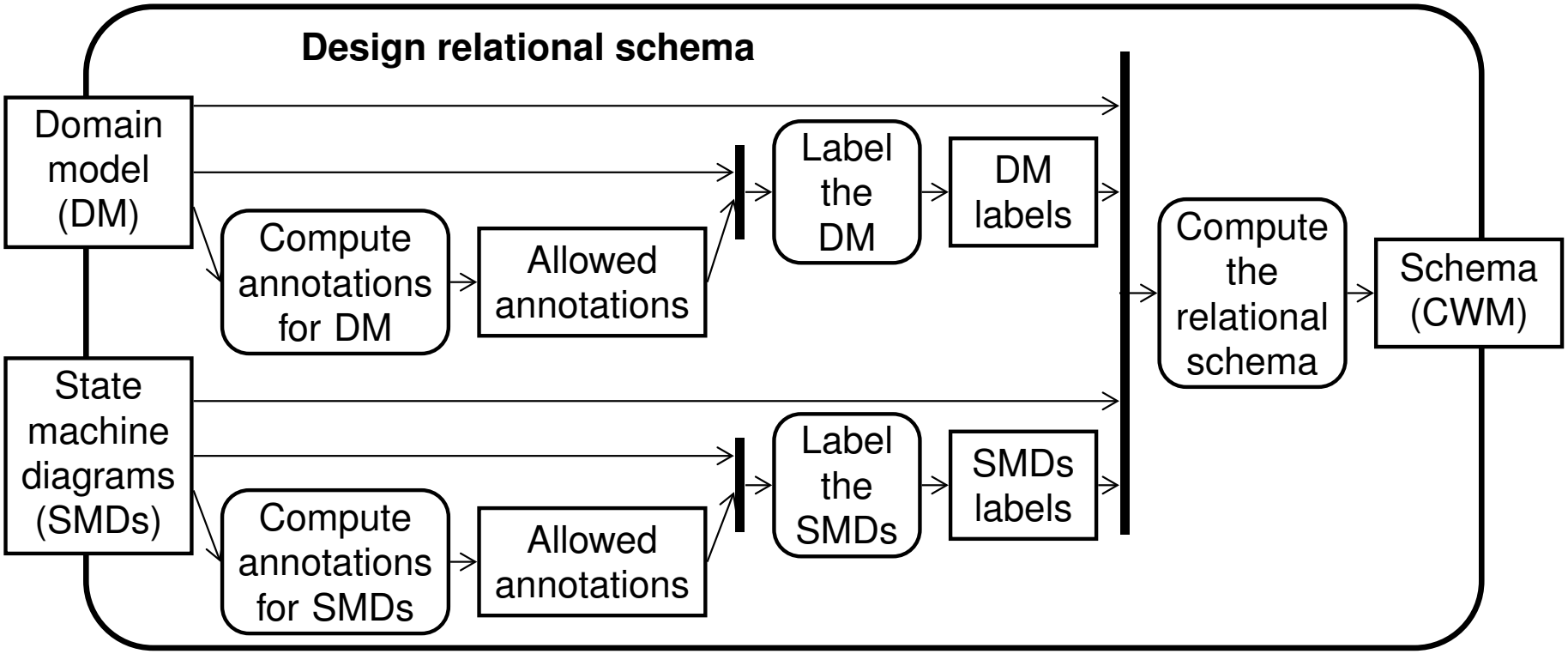
Problem statement 1: Tackle the impedance mismatch



Problem statement 2: Integrate MDE technologies into a design tool

- Harness the infrastructure complexity (languages, metamodels, xmi, development tools)
- Develop and debug the transformations
- Put it all together (models transformations execution, additional functionalities, graphical interface)

Design process activity diagram

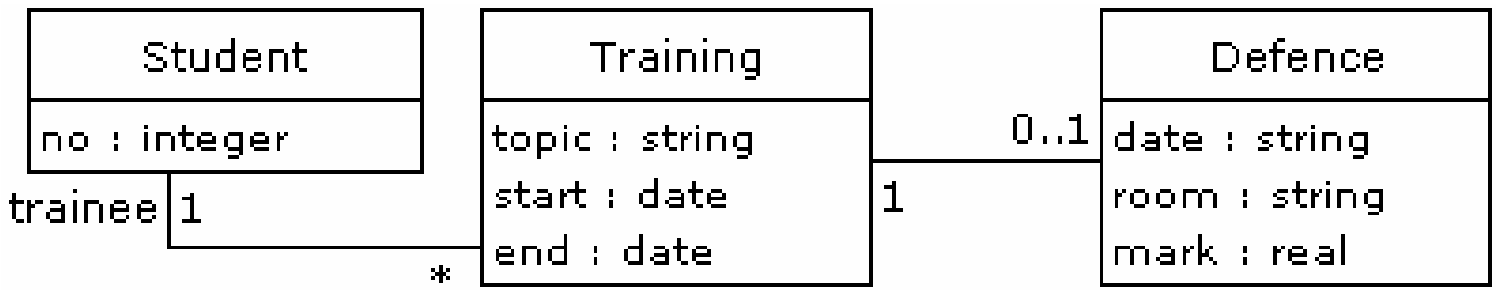


The Domain Model (DM)

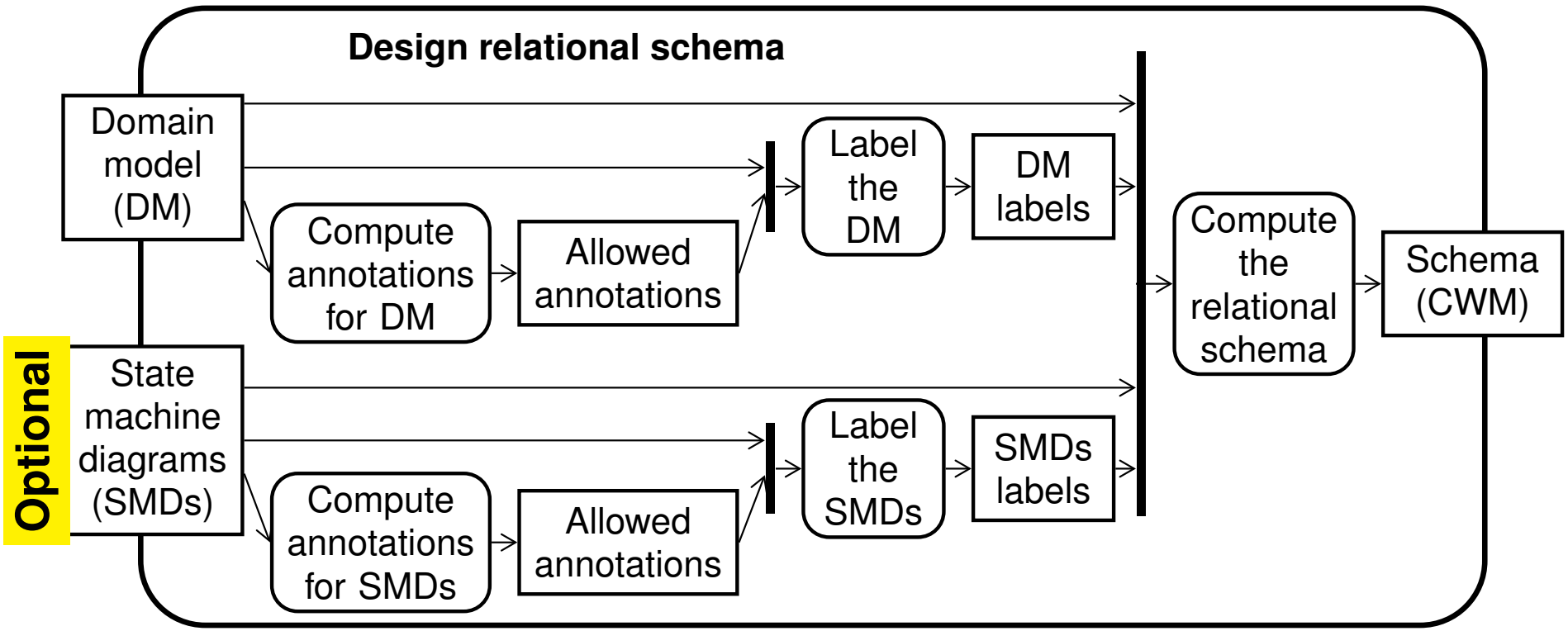
The main input of the process is a domain model

- It is the central model during IS development (here represented by a UML class diagram)
- It is not an application's object model

Input domain model example



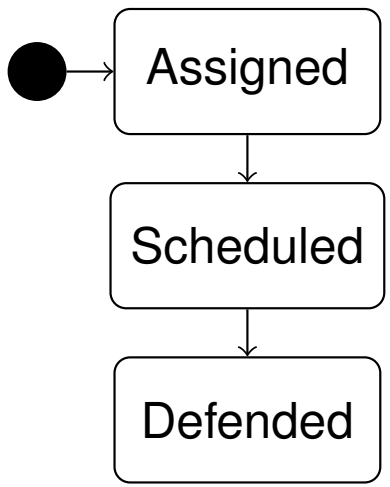
Behavioral State Machine Diagrams (SMDs)



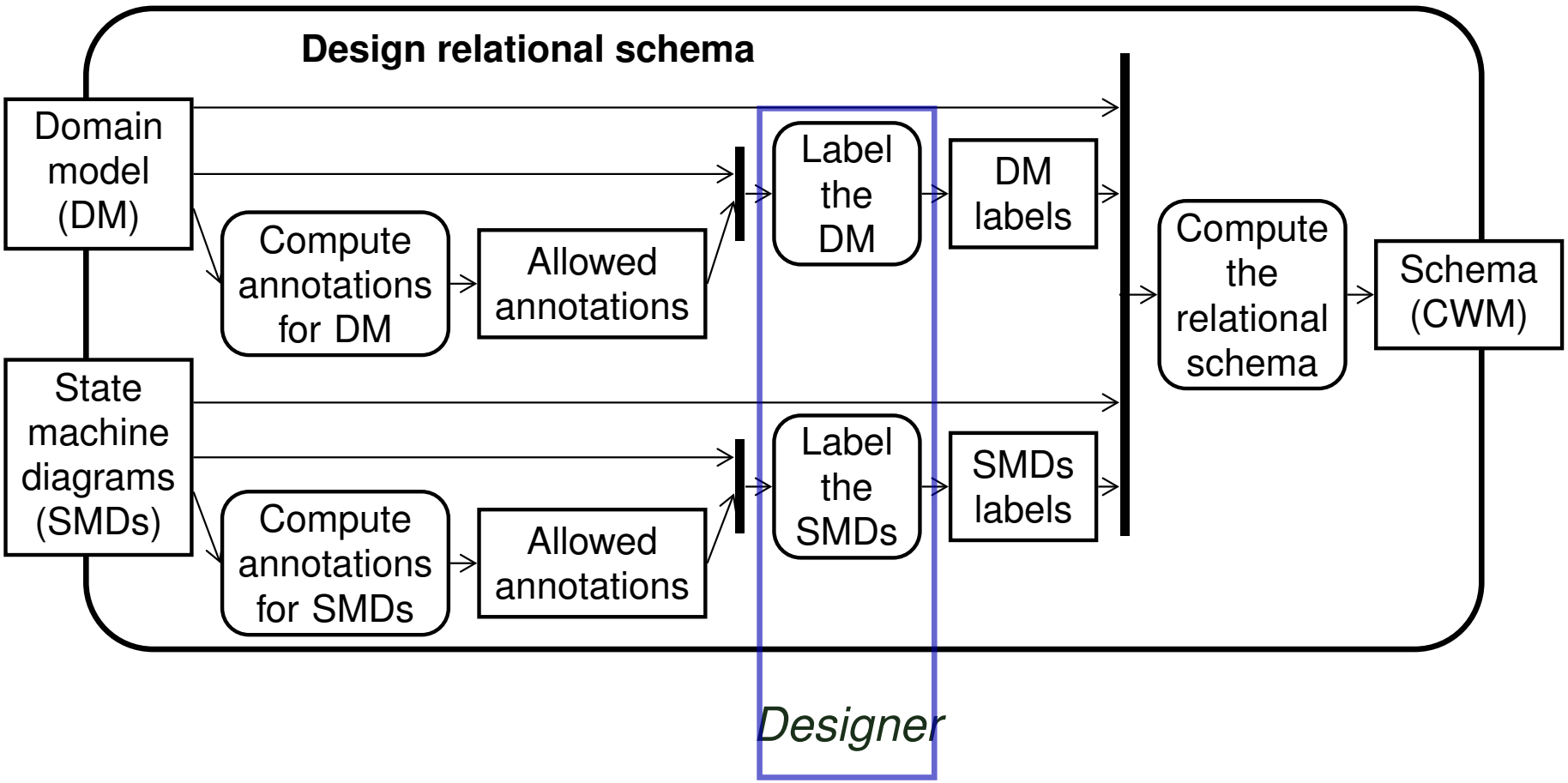
Behavioral State Machine Diagrams (SMDs)

They provide additional information on the life cycle of the DM classes

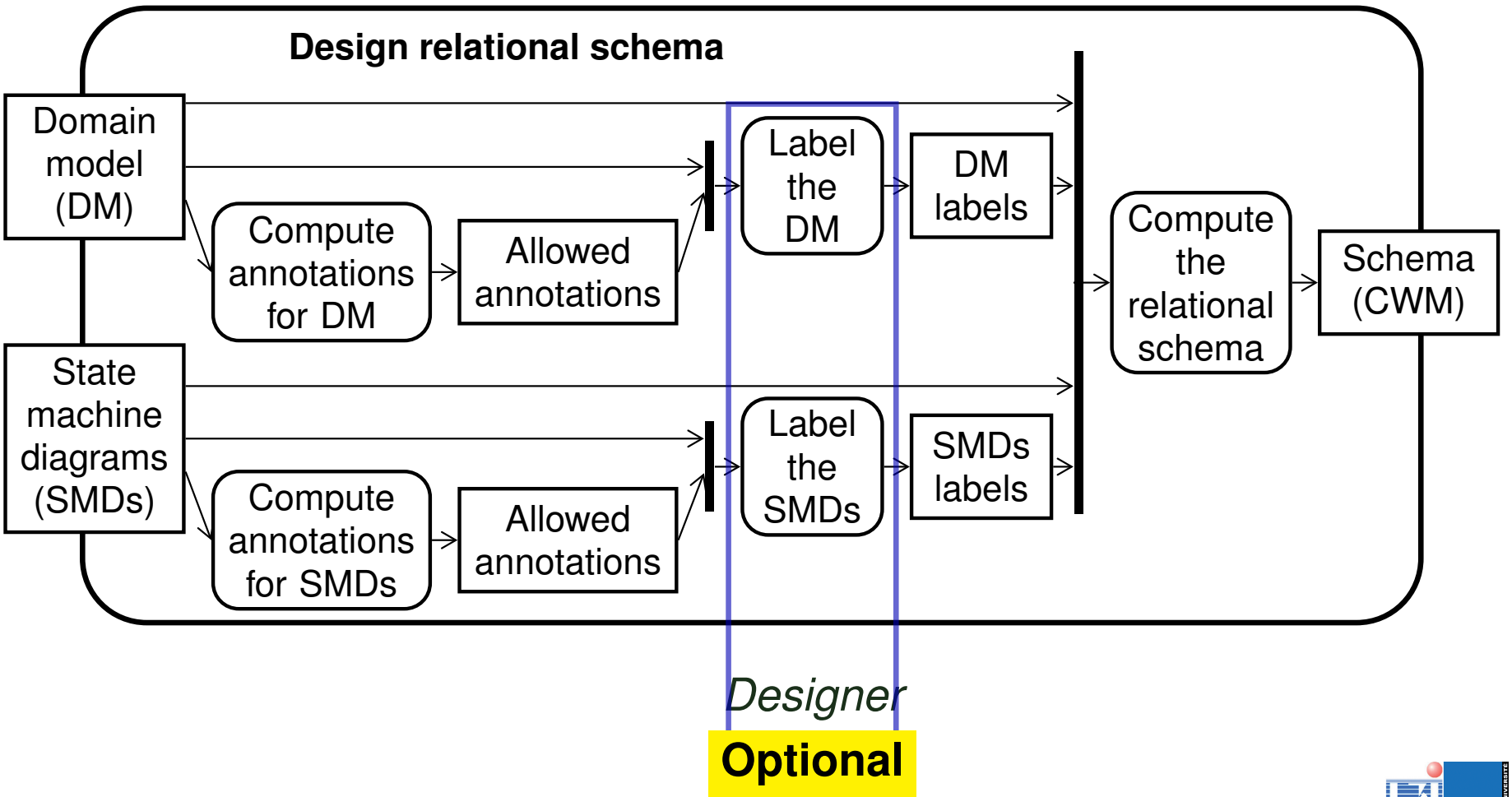
State machine diagram example for the `Training` domain class (very simple indeed)



Annotations on the DM



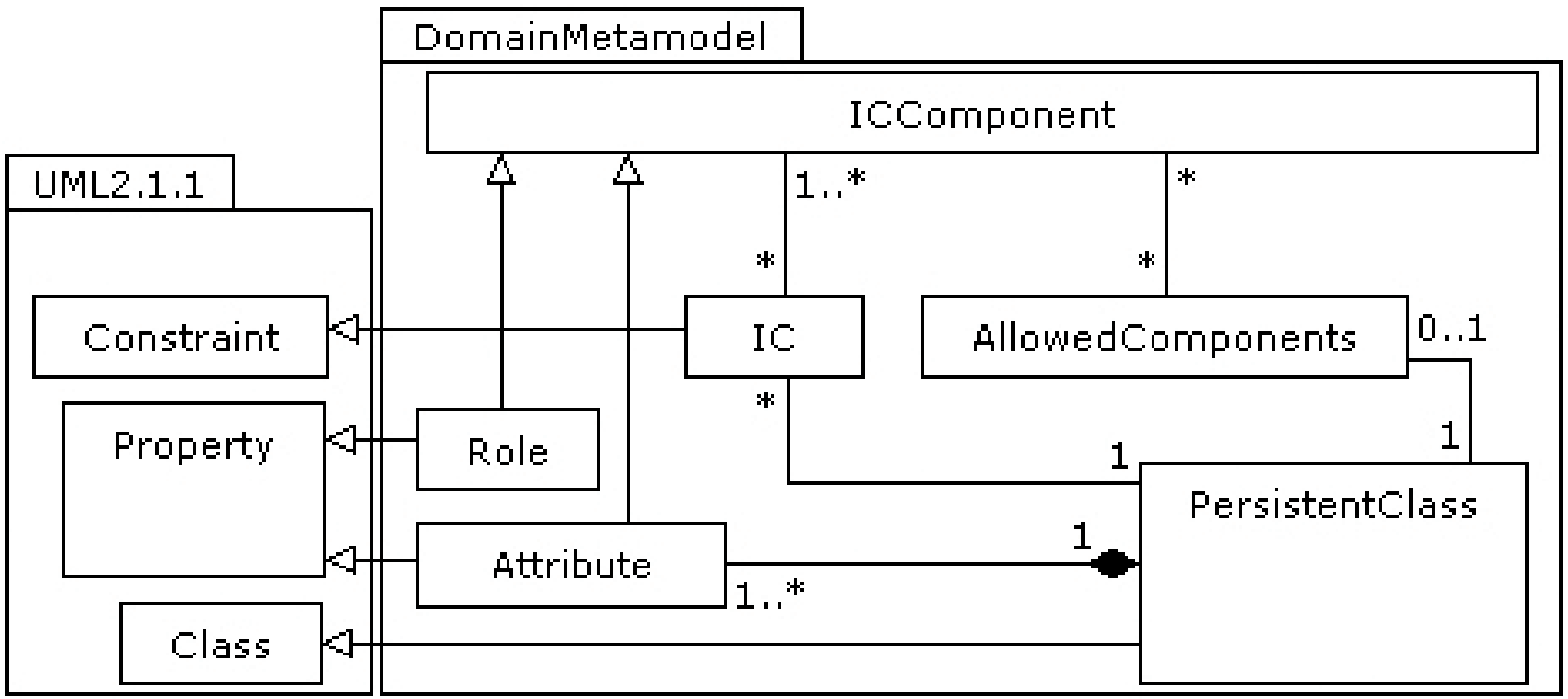
Annotations on the DM



Annotations on the DM

Relational unique constraints (keys) are essential constructs to enforce data consistency. We propose to express "keys" on the DM as UML Constraints, using ad-hoc stereotype: Identification Constraints (IC)

The metamodel for Identification Constraints



Supervision

We supervise the designer during Identification Constraints construction

1. For each persistent class, the system computes a set of allowed "ICComponents", i.e. components that may compose an Identification Constraint for the class (the computation takes generalizations and association multiplicities in account)
2. The designer selects ICComponents from the computed set
3. The system checks identification constraints validity

Supervision

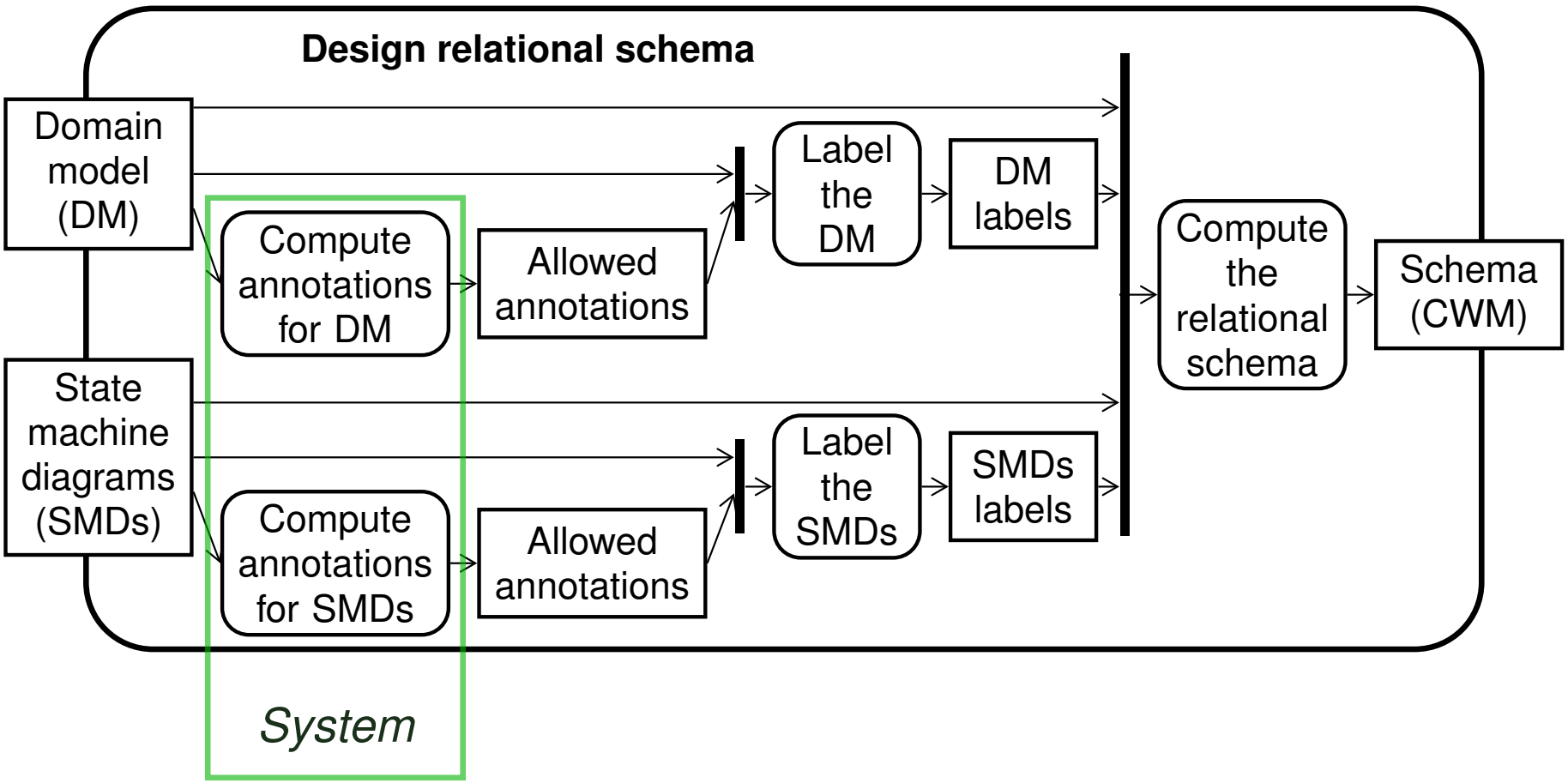
We supervise the designer during Identification Constraints construction

1. For each persistent class, the system computes a set of allowed "ICComponents", i.e. components that may compose an Identification Constraint for the class (the computation takes generalizations and association multiplicities in account)
2. The designer selects ICComponents from the computed set
3. The system checks identification constraints validity

Several benefits

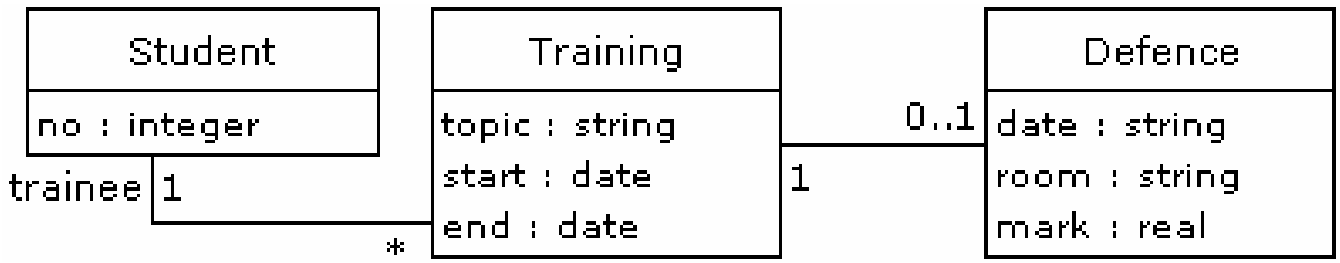
- The designer might discover unexpected ICComponents (not only class attributes, but also roles)
- Quality of the resulting relational unique constraints

Supervision



Examples

The DM

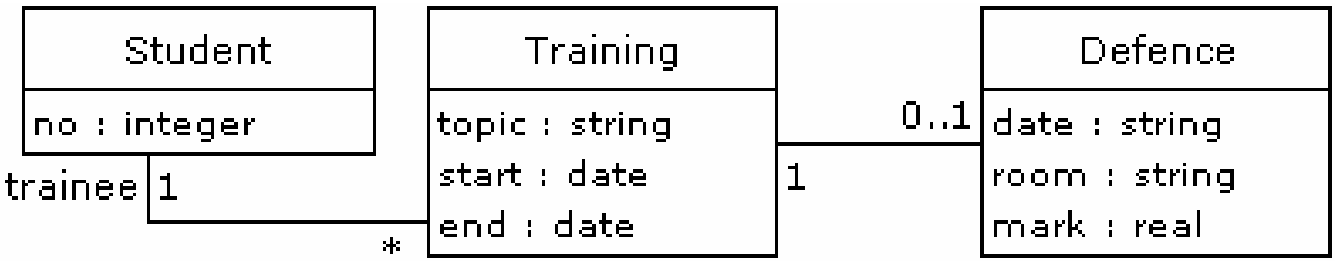


For the Training class, the system determines that the following identification constraint components may be used by the designer

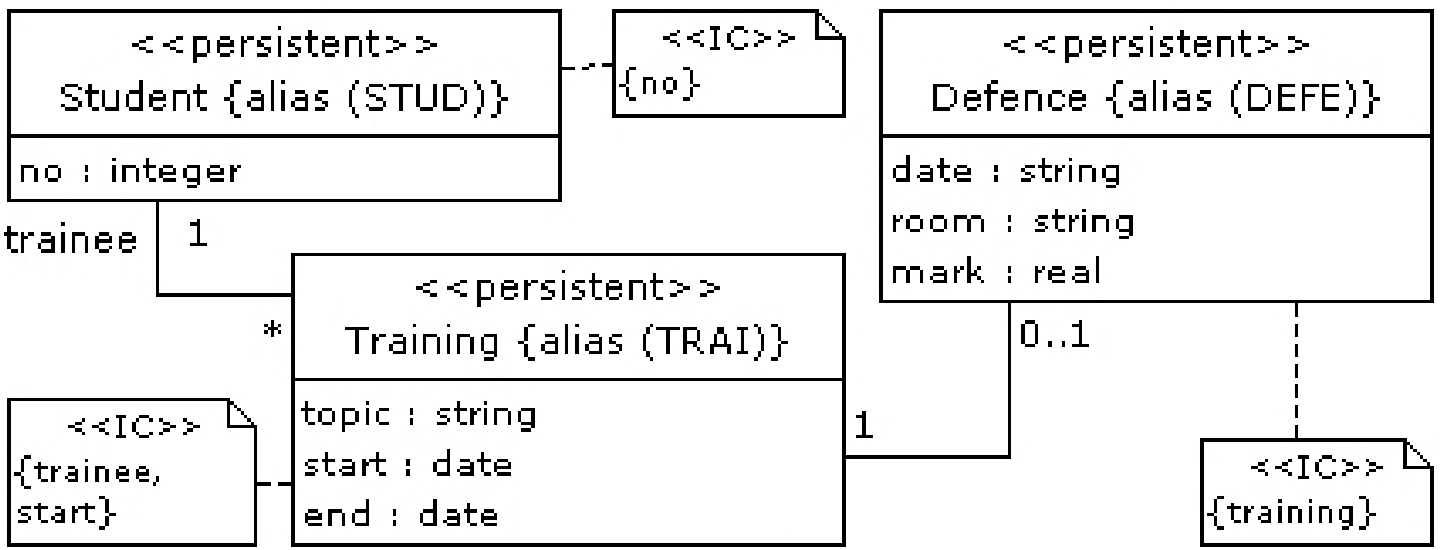
attributes: `topic, start, end`
roles: `defence, trainee`

Examples

The DM



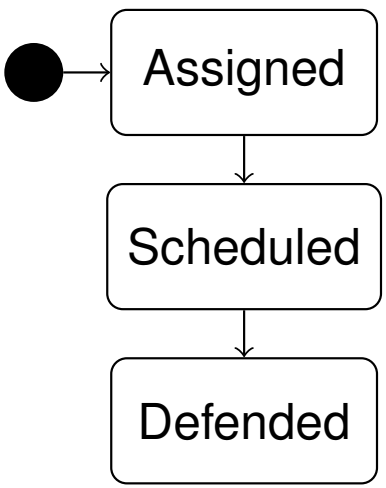
The DM with correct annotations added by the designer



Annotations on the SMDs

Additional state constraints are useful to process, to ensure further data consistency. We propose to allow the designer to add state constraints on SMDs

Example of labels added by the designer on the SMD

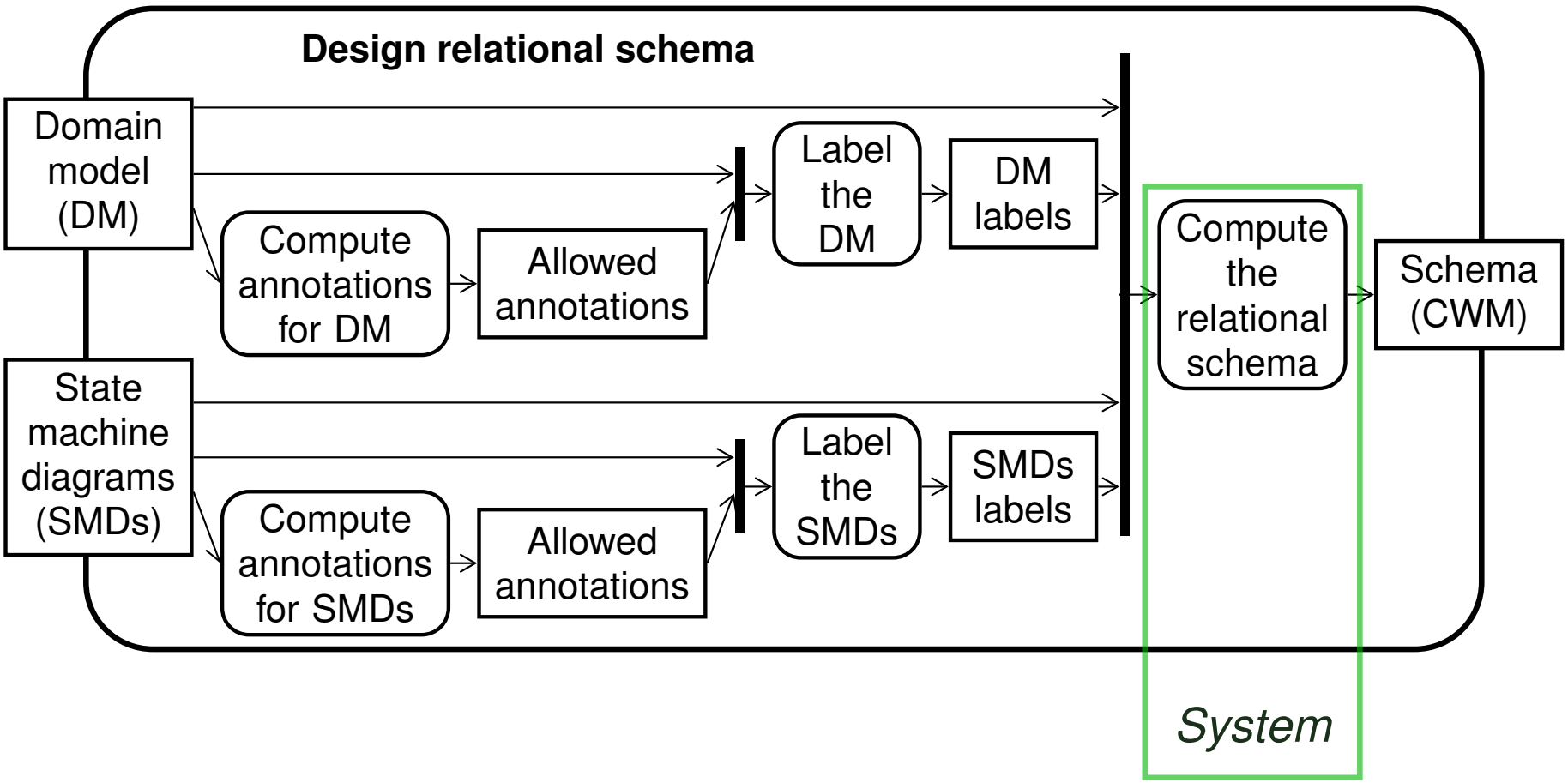


$\Leftrightarrow$ There is no link between *Training* and *Defence* instances

$\Leftrightarrow$ There is a link between *Training* and *Defence* instances

$\Leftrightarrow$ There is a link between *Training* and *Defence* instances AND *Defence* is in *"marked"* state

Crossing the barrier



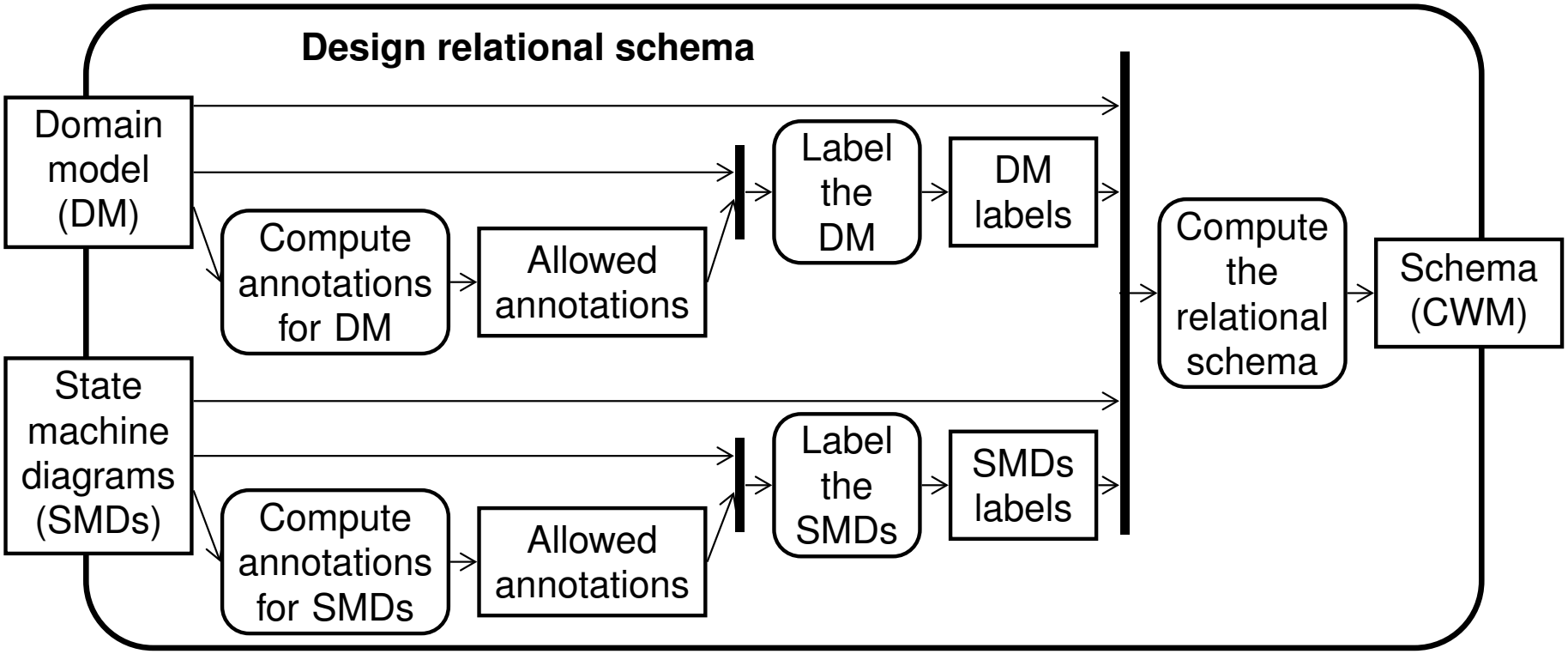
Crossing the barrier

Based on the DM, the SMDs and their respective annotations, a QVT transformation computes the relational schema.

Key features of the transformation

- An association is processed according to its multiplicities to become Attribute(s) and Foreign Key(s) and optional Table and Unique Constraint(s)
- A Generalization yields Foreign Key and Unique Constraint
- An Identification Constraint yields Unique Constraint
- A State Constraint yields TRIGGER and CHECK
- Names (classes, attributes, roles, associations) are preserved everywhere it's syntactically possible (SQL syntax)

Relational schema



Relational schema

The output relational schema is a logical schema (ANSI/X3/SPARC/DBS-SG acceptation), there are no optimization for efficiency: it is not a physical schema

Logical schema

- `CREATE TABLE`
- `ADD CONSTRAINT`
- `TRIGGER`
- `CHECK`

Physical schema

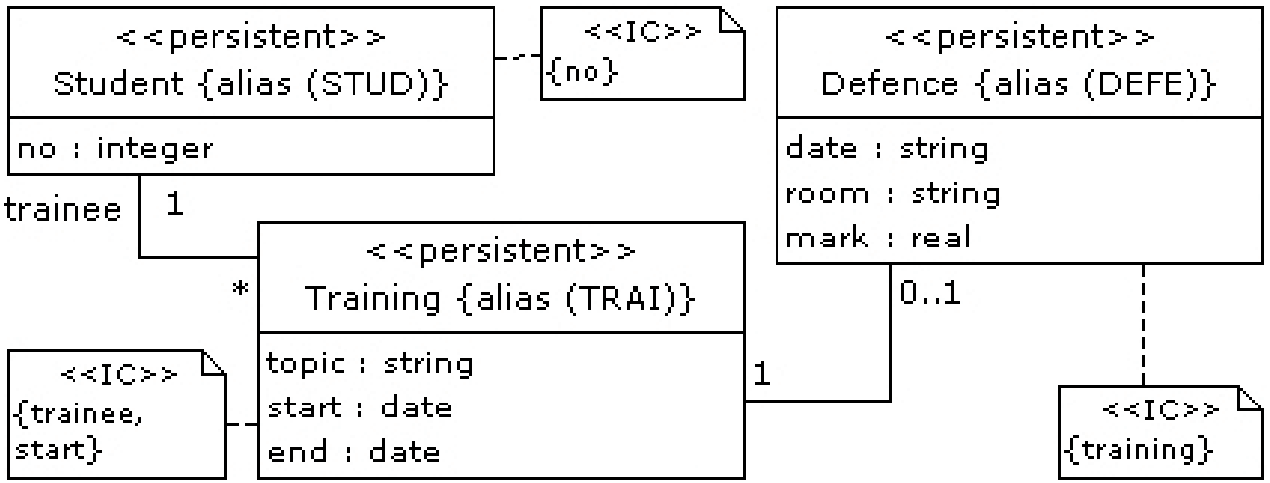
- `CREATE TABLESPACE`
- `CREATE INDEX`
- `...`

CWM (Common Warehouse Metamodel)

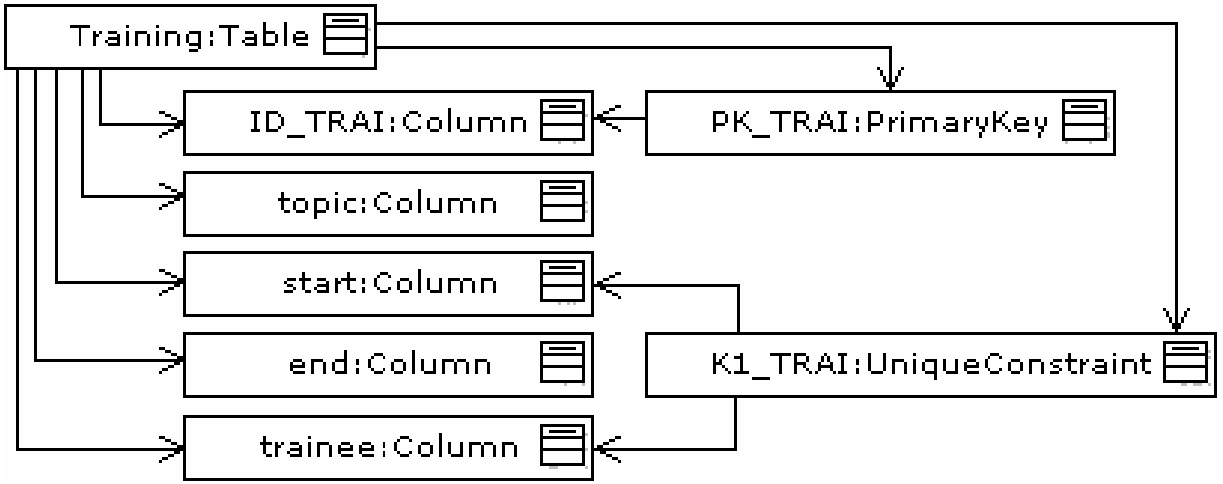
Management	Warehouse Process			Warehouse Operation		
Analysis	Transformation		OLAP	Data Mining	Information Visualization	Business Nomenclature
Resource	Object Model	Relational	Record	Multidimensional		XML
Foundation	Business Information	Data Types	Expression	Keys and Indexes	Type Mapping	Software Deployment
	Object Model					

- a metamodel for datawarehouse metadata interchange (not limited to relational)
- MOF compliant, fits in MDA
- used by existing CASE tools to generate SQL code

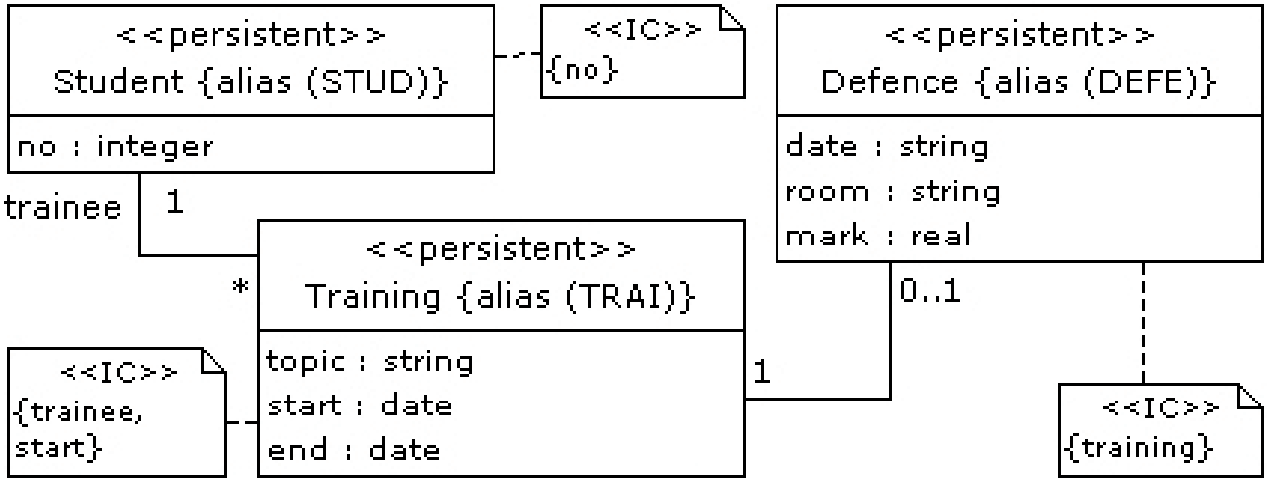
The result



Output CWM relational model (exerpt). Very cumbersome...



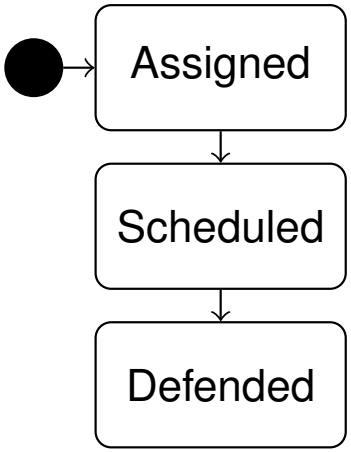
The result



SQL code of the output CWM relational model (exerpt)

```
CREATE TABLE Training (
    ID_TRAI   NUMBER    NOT NULL, topic VARCHAR(30) NOT NULL,
    start     DATE      NOT NULL, end   DATE        NOT NULL,
    trainee   NUMBER    NOT NULL,
  CONSTRAINT PK_TRAI PRIMARY_KEY (ID_TRAI),
  CONSTRAINT K1_TRAI UNIQUE (trainee,start));
ALTER TABLE Training ADD CONSTRAINT FK_STUD FOREIGN_KEY (Trainee)
    REFERENCES Student.ID_STUD;
```

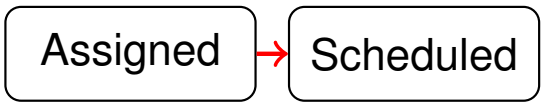
The result



SQL code of the output CWM relational model (exerpt)

```
CREATE TABLE Training (
    ID_TRAI  NUMBER   NOT NULL, topic VARCHAR(30) NOT NULL,
    start    DATE     NOT NULL, end   DATE        NOT NULL,
    trainee  NUMBER   NOT NULL,
    state_trai VARCHAR(15) NOT NULL,
  CONSTRAINT PK_TRAI PRIMARY_KEY (ID_TRAI),
  CONSTRAINT PK_TRAI PRIMARY_KEY (ID_TRAI),
  CONSTRAINT K1_TRAI UNIQUE (trainee,start),
  CONSTRAINT chk_state_trai
    CHECK (state_trai IN ('Assigned', 'Scheduled', 'Defended')));
ALTER TABLE Training ADD CONSTRAINT FK_STUD FOREIGN_KEY (Trainee)
    REFERENCES Student.ID_STUD;
```

The result

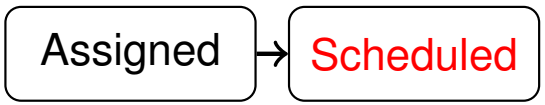


$\Leftrightarrow$ There is a link between *Training* and *Defence* instances

SQL code of the TRIGGER for the `Scheduled` state

```
create or replace trigger trig_det_training
before inserting or update on training
for each row
declare verif_exists_defence number;
begin
 select count(*) into verif_exists_defence
 from defence d where :new.id_trai = d.training;
 if inserting
  then...
  elseif updating
   then
    if ( :new.state_trai = 'Scheduled'
     and :old.state_trai <> 'Assigned')
     then raise_application_error() ;
     elseif ( :new.state_trai = 'Scheduled'
      and verif_exists_defence = 0)
      then raise_application_error() ;
     end if ; end if ; end ;
```

The result



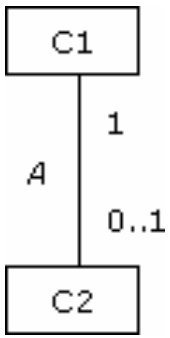
$\Leftrightarrow$ There is a link between *Training* and *Defence* instances

SQL code of the TRIGGER for the `Scheduled` state

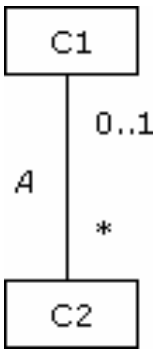
```
create or replace trigger trig_det_training
before inserting or update on training
for each row
declare verif_exists_defence number;
begin
 select count(*) into verif_exists_defence
 from defence d where :new.id_trai = d.training;
 if inserting
  then...
  elseif updating
   then
     if ( :new.state_trai = 'Scheduled'
      and :old.state_trai <> 'Assigned')
      then raise_application_error() ;
     elseif ( :new.state_trai = 'Scheduled'
      and verif_exists_defence = 0)
      then raise_application_error() ;
     end if ; end if ; end ;
```

A few examples

Associations



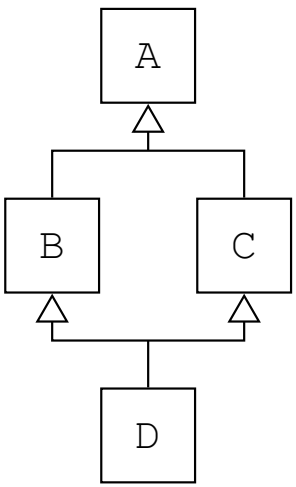

```
CREATE TABLE C1(
  ID_C1NUMBER  NOT NULL PRIMARY_KEY);
CREATE TABLE C2 (
  c1    NUMBER  NOT NULL UNIQUE
        REFERENCES C1.ID_C1);
```




```
CREATE TABLE C1(
  ID_C1NUMBER  NOT NULL PRIMARY_KEY);
CREATE TABLE C2 (
  c1    NUMBER  NULL     UNIQUE
        REFERENCES C1.ID_C1);
```

A few examples

Generalizations




```
CREATE TABLE A(
  ID_A NUMBER NOT NULL PRIMARY_KEY);
CREATE TABLE B (
  ID_B NUMBER NOT NULL PRIMARY_KEY,
  A    NUMBER NOT NULL UNIQUE
       REFERENCES A.ID_A);
CREATE TABLE C (
  ID_C NUMBER NOT NULL PRIMARY_KEY,
  A    NUMBER NOT NULL UNIQUE
       REFERENCES A.ID_A);
CREATE TABLE D (
  B    NUMBER NOT NULL UNIQUE
       REFERENCES B.ID_B,
  C    NUMBER NOT NULL UNIQUE
       REFERENCES C.ID_C);
```

Implementations

- ATL experiment
  - Several transformations (allowed annotation computing, object models relational translation)
  - About 25 mapping rules (depending on the way we implement the process)

ICToUniqueConstraint ATL rule

```
rule ICToUniqueConstraint extends ElementToModelElement {
  from   e : UML!IC
  using {   class : UML!PersistentClass = e.owner;
            table : CWM!Table = thisModule.resolveTemp(class, 'm');    }
  to    m : CWM!UniqueConstraint (  name <- table.UNQName(class.alias),
        feature <- e.composition
           ->select( ic_component | ic_component.oclIsTypeOf(UML!Attribute) )
           ->union( thisModule.importedRolesByID(e) )->flatten()   )
  do {   table.ownedElement_UC <- m;  table.UNQCounter() <- table.UNQCounter() + 1;
  }
}
```

Implementations

- ATL experiment
  - Several transformations (allowed annotation computing, object models relational translation)
  - About 25 mapping rules (depending on the way we implement the process)
- Python experiment

QVT (declarative) vs. imperative programming language

- QVT: straightforward to implement simple transformations
- Imperative: easier to control complex transformation and interface with user events and GUI widgets

Implementations

- ATL experiment
  - Several transformations (allowed annotation computing, object models relational translation)
  - About 25 mapping rules (depending on the way we implement the process)
- Python experiment
- ongoing CASE tool integration:
  We are developing an Eclipse plugin, based on the UML2 Tools plugin which provides class diagram and state machine diagram graphical editors

Formal proof of the process output quality

Distance to normal forms

- close to Boyce-Codd normal form ?
- we don't have functional dependencies at hand
- classical normalization does not take NULL into account

Validation

- multiple test-cases
- need for complete integration

Conclusion

Expected contributions (to the MODELS community)

- Some other problem statements! (Intermodel references handling...)
- A comparison of model transformation languages usability
- Teachings from the complete integration of MDE in a design tool (userfriendliness, tooling...)

Thank you