# Designing relational schema from an object model using model transformations

Antoine Wiedemann

# Designing relational schema from an object model using model transformations

Antoine Wiedemann[1]
awiede01@univ-lr.fr

SIDO, L3i, Université de La Rochelle

## 1 Introduction: Problem statement

The domain model [12, 8] hereafter referred to as DM is a central model during information systems development. It serves to visualize *real-world* concepts as opposed to *software* objects. We want to help the database designer to derive from the DM a relational structure eventually made available to potentially multiple applications.

But there is no direct mapping between concepts of object and relational models, and as shown in table 1, there is a so-called "object-relational impedance mismatch" [4]. For example the internal identification of objects is very different from the identification of tuples. A relational tuple is identified by the value of a subset of its attributes (table columns). Each of the subsets is a key, and is defined by a `unique` constraint. There may be more than one key in a table. Integrity constraints (unique -including primary key-, referential and check constraints) are an important part of every quality relational schema, as they prevent storing in its tables data that would be inconsistent with the domain model.

| UML concepts | | Relational concepts |
|---|---|---|
| Class | → | Table |
| Association | → | |
| Generalization | → | Foreign Key (referential constraint) |
| Attribute | → | Column |
| Role | → | |
| Constraint | ← | Key (unique constraint) |
| [0..1] multiplicity | → | no not null constraint |

**Table 1.** The object relational impedence mismatch

Previous approaches use E/R diagrams [5] as a design language to produce a relational schema or SQL code. However, they do not handle [0..1] multiplicities and do not generate SQL code with NULL values. Moreover today the UML is widely accepted and used so we try to keep the database designer work in the same formalism. Many patterns are defined in the literature to map UML to relational concepts, but the transformation is not automated [18, 7]. Different

CASE tools enable relational model generation from an UML diagram, but the quality of the result is not satisfactory [11]: missing unique constraints, no way to use role to express constraints. Besides relational database design, some patterns consider the mapping between object and relational concepts [1]. Some are implemented in various Object Relational Mapping (ORM) tools that provide a persistence layer to applications objects and hide the underlying database architecture (Hibernate, JDO, ActiveRecord[1], ...). Constraints checking is then mainly left to the applications while the RDBMS capabilities are under-used. So mapping Java objects to relational tables and automatically generating queries is not the problem we try to solve. We do not provide an ORM persistence layer that would hide the RDBMS and we do not automatically generate queries [13].

## 2 Our proposal

We propose an automated relational database design method starting from a DM. This DM is assumed correct and we do not consider its validation. We provide specific annotations enabling the designer to express identification constraints (IC) on DM objects. An object IC may be composed of attributes and/or roles. Each IC is bound to become a unique constraint in the relational schema. We help the designer during the DM annotation phase, by computing sets of allowed IC components. From the annotated DM we derive a relational model with integrity constraints. We believe that the targeted relational model avoids redundancy although classical normalization theory [6] doesn't apply, since it does not take null values into account. Classical normalization imposes classes merging in case of 1-1 association. To respect the designer vocabulary more closely, we sometimes step back from these principles. The relational model produced by our process can be viewed as a multi-purpose model from which relational "artefacts" (UML 2.0) can be derived (SQL, physical schema ...).

### 2.1 Implementation using MDE

To implement our process, we use model driven engineering (MDE) and more precisely the MDA[16] approach. Working in the MDA environment, we naturally use CWM metamodel [15] to instantiate our output relational model, which makes the process robust to SQL variants. CWM is an OMG specification for modeling metadata of a datawarehousing environment, including relational constructs. To be fully OMG-compatible, until now we have used a QVT-like transformation model language [17, 3].

Figure 1 is a short example of a sample input DM. In this domain, students are assigned trainings which are eventually defended. A student may be assigned one or more trainings, while a training can only be performed by one student. A defence always belongs to a single training whereas a training may or may not have its defence already accomplished.

[1] http://www.hibernate.org, http://java.sun.com/jdo/, http://wiki.rubyonrails.org/rails/pages/ActiveRecord

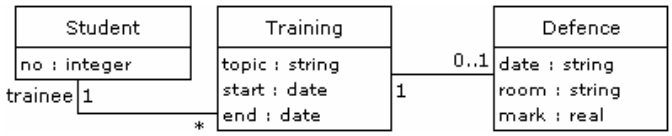


**Fig. 1.** Sample DM

Out of this DM the designer is first asked to tag persistent classes. The selected classes are stereotyped «persistent» and are represented as PersistentClass instances of our UML extension presented in Figure 2. Given the new persistent tagged DM and its multiplicities, an automatic QVT transformation selects for every persistent class the set of possible IC components as Attribute or Role instances. These results are added to the DM as $A_i$ instances of the

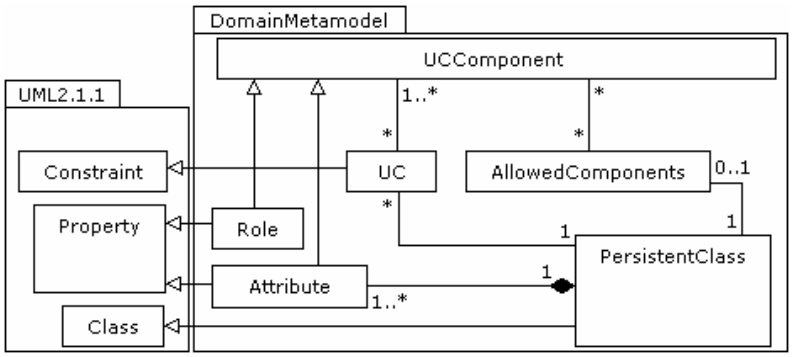


**Fig. 2.** Extended UML

AllowedComponents metaclass and links between $A_i$ and selected ICComponent instances. Automatically restricted sets of components are next proposed to the designer, thus ensured to build correct identification constraints as IC instances. Again, these constraints are represented using a stereotype with tagged values that encode the designer construction. In Figure 3 example, ICs include both attributes (no, start) and roles (trainee, training[2]). We do not use OCL: its great expressiveness is not needed here, and we try to keep annotations as light and user-friendly as possible.

Based on this last input, another QVT transformation fully automates the critical mutation of the annotated DM into its CWM-compliant relational form, hence crossing the object/relational barrier. Instead of the CWM model, Figure 4 shows the corresponding SQL script. There are no difficulties to match CWM constructs and SQL keywords, and existing CASE tools [11] successfully implement this translation. The SQL code not only creates tables but also adds

[2] The class name of Training is used as a role in the Training/Defence association

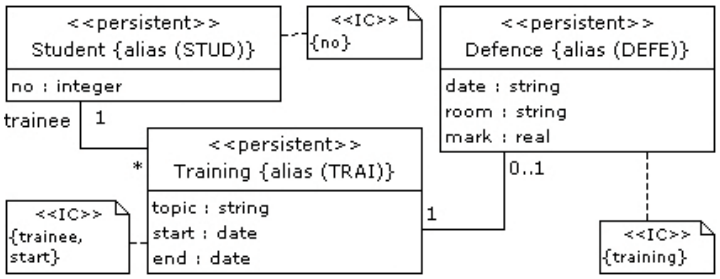


**Fig. 3.** DM with designer annotations

constraints to the schema. Italicized items were derived from the designer annotations, and could not have been inferred otherwise.

```
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;
```

**Fig. 4.** SQL of the output relational model

### 2.2 Problems encountered so far

Using the QVT approach, we build transformations that take a whole model as input. This makes the input model refactoring costly, since even a small change on it forces to recompute everything from the start. Perhaps we should split our transformations into smaller ones.

Another challenge we face is the definition of a mechanism suited to practically add annotations to our DM. At the time of writing we do not have a fully integrated CASE tool that would take care of the translation of graphical symbols (stereotypes) into metaclasses instances. To support the formal proof that our relational schema avoids redundancy, we also try to generate on the fly traceability links between input and output models.

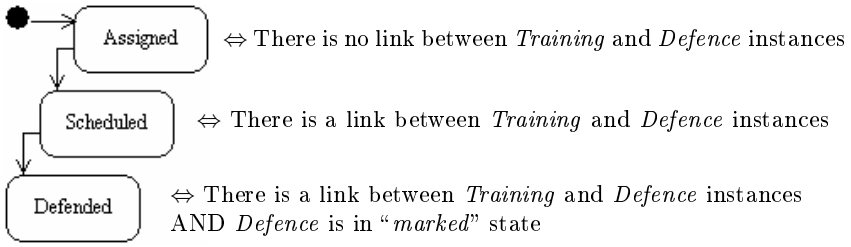


**Fig. 5.** Sample state machine diagram of the `Training` class and additional designer annotations

### 2.3 Taking the dynamic model into account

En plus de la partie statique du modèle du domaine, nous cherchons à traiter en même temps les informations que nous donne le concepteur sur la dynamique des classes, exprimée par des diagrammes états/transition (DET). De cette manière, en récupérant des contraintes supplémentaires, nous voulons obtenir un schéma relationnel encore plus fidèle au modèle du domaine.

More than a static structural model [2, 14], we strive to process at the same time the information of behavioral state machines diagram (SMD) with concurrent states. This way, gathering more constraints, we expect to get relational models even more consistent with the domain. Figure 5 shows an example SMD for the Training class of the DM depicted in Figure 1. The processing of this SMD leads to the creation of a specific "state" attribute in the corresponding Training table, whose value is either Assigned, Scheduled or Defended. In the same way as we did on the DM, we want to assist the designer during another annotation step on the SMD. An example of these annotations are shown on the right hand side of Figure 5. These annotations are more complex to implement and control than the ones on the DM, since they combine several models, namely the current SMD, the DM, and potentially other SMD. Italicized words in Figure 5 refers to these foreign elements. We add CHECK and TRIGGERS to the relational model according to these inputs. Working with two or more models at the same time is diving into model composition [9].

## 3 Contribution

The whole process tends to ensure the quality of the resulting schema with the same purpose as classical normalization [6], i.e. minimizing redundancy, which is very different from ORM techniques. We decide to retain - within the database schema - the meaningful names from the domain terminology, especially by avoiding some inopportune table fusions or creations. We provide support to the designer through control and supervision during the annotation process on DM and SMDs, enabling further automation of the design process. Contrary to E/R-like methods, we handle 0..1 multiplicities in the DM and accept NULL constraints in the relational model.

With respect to MDE implementation, we extend UML by defining appropriate specializations of some of its metaclasses. Therefore we take benefit of the stereotypes extension mechanism, adopt their standard notation and make the various user-annotation steps fully UML-compliant. As outlined in Section 2.2, our work raises various questions about MDA integration. Beyond the simple MDA transformation pattern of a single model into another, we also have to master more complex model processing, such as model and inter-model annotation and also model composition. We expect to bring new contributions by coping with these challenges.

## References