Model Validation in Ontology Based Transformations*

Jesús M. Almendros-Jiménez
jalmen@ual.es
Dpto. de Lenguajes y Computación
Universidad de Almería
04120-Spain

Luis Iribarne
luis.iribarne@ual.es
Dpto. de Lenguajes y Computación
Universidad de Almería
04120-Spain

Model Driven Engineering (MDE) is an emerging approach of software engineering. MDE emphasizes the construction of models from which the implementation should be derived by applying model transformations. The Ontology Definition Meta-model (ODM) has been proposed as a profile for UML models of the Web Ontology Language (OWL). In this context, transformations of UML models can be mapped into ODM/OWL transformations. On the other hand, model validation is a crucial task in model transformation. Meta-modeling permits to give a syntactic structure to source and target models. However, semantic requirements have to be imposed on source and target models. A given transformation will be sound when source and target models fulfill the syntactic and semantic requirements. In this paper, we present an approach for model validation in ODM based transformations. Adopting a logic programming based transformational approach we will show how it is possible to transform and validate models. Properties to be validated range from structural and semantic requirements of models (pre and post conditions) to properties of the transformation (invariants). The approach has been applied to a well-known example of model transformation: the Entity-Relationship (ER) to Relational Model (RM) transformation.

1 Introduction

Model Driven Engineering (MDE) is an emerging approach for software development. MDE emphasizes the construction of models from which the implementation should be derived by applying model transformations. Hence, the *model transformation* [29, 16] is a key tool of MDE. According to the *Model Driven Architecture (MDA)* [23] initiative of the *Object Management Group (OMG)* [22], the model transformation provides to developers a framework for transforming their models.

The MDA approach proposes (at least) three levels in order to describe a model transformation: the first one is the so-called *meta-meta-model*, which is the basis of the model transformation, and provides the language for describing meta-models. The second one consists in the *meta-models* of the models to be transformed. Source and target models must conform to the corresponding meta-model. Such meta-models are modeled according to the meta-meta-model. The third one consists in the source and target *models*. Source and target models are instances of the corresponding meta-models. In addition, source and target meta-models are instances of the meta-meta-model. In order to define a model transformation one should be able to meta-model the source and target models with regard to the meta-meta-model, and map source and target meta-models. Model transformation needs formal techniques for specifying the transformation. In most cases transformations can be expressed in some kind of *rules*.

On the other hand, the *Ontology Definition Metamodel (ODM)* proposal [25] of the *OMG* aims to define an ontology-based representation of UML models. ODM is a standard for representing UML

*This work has been supported by the Spanish Ministry MICINN and Ingenieros Alborada IDI under grant TRA2009-0309. This work has been also supported by the EU (FEDER) and the Spanish Ministry MICINN under grants TIN2010-15588, TIN2008-06622-C03-03, and the JUNTA ANDALUCIA (proyecto de excelencia) ref. TIC-6114.

J. Silva and and F. Tiezzi (Eds.): Workshop on Automated Specification and Verification of Web Systems (WWV 2012)
EPTCS 98, 2012, pp. 17–30, doi:10.4204/EPTCS.98.4

models by OWL in which, among others, UML classes are mapped into ontology concepts, UML associations are mapped into ontology roles, and multiplicity restrictions of UML are mapped into cardinality restrictions in roles. ODM is itself an UML meta-model in which UML models can be accommodated. Following the ODM proposal, an UML model can be represented by an ontology in which the **TBox** (i.e. the *terminological box*) contains the UML meta-model while the **ABox** (i.e. the *assertional box*) contains the instance of the UML meta-model which represents the model.

Model validation is a key element of MDE. Firstly, (a) source and target models must conform to the corresponding meta-models. Source and target meta-models describe the *syntactic structure* of source and target models. However, some *semantic requirements* have to be imposed on source and target models. In UML semantic requirements are usually expressed in the *Object Constraint Language (OCL)* [24]. Secondly, (b) *pre-conditions* and *post-conditions* and *invariants* are imposed on transformations. While source and target models can be well-formed with regard to meta-models, some extra requirements can be required. We can distinguish two specific cases: (b.1) *source and target model requirements*, and (b.2) *transformation requirements*. The first case covers requirements of source and target models in isolation. The second case covers requirements on target models with regard to the source models.

In the ODM context, one can argue that the use of OWL as modeling language provides a suitable framework for validation of properties. OCL can be replaced by OWL when specifying requirements imposed on models. *OWL reasoning* is a widely studied topic of research, and many tools have been developed in this context (for instance, the *Protégé* tool [18] and the OWL reasoners *Hermit*, *Jena*, *Fact++*, *Racer*, among others). OWL reasoning ranges from *ontology consistence testing* to *ontology-based inference* (i.e. derivation from ontology axioms). The topic can be applied to ODM (an hence to UML) however, validation in model transformation is a wider topic of research. Model validation involves ontology consistence testing and ontology-based inference for cases (a) and (b.1), while case (b.2) involves cross validation of ontologies.

On the other hand, the relationship between logic programming and ontologies is well-known. OWL is based on the *Description Logic (DL)* [5], a family of fragments of *first order logic*, and some DL fragments can be encoded into logic programming, for instance, the so-called *Description Logic Programming* approach [12], and OWL RL [21, 3]. Typically, Description Logic is used for representing the **TBox** and the **ABox**. The encoding of (fragments of) DL into logic programming is based on the representation of the **TBox** by Prolog rules and the representation of the **ABox** by Prolog facts. It means that ontology instances are represented by Prolog facts.

In this paper, we present an approach for model validation in ODM based transformations. Adopting a logic programming based transformational approach we will show how it is possible to transform and validate models. Properties to be validated range from structural and semantic requirements of models (pre and post conditions) to properties of the transformation (invariants). The approach has been applied to the well-known example of model transformation: the Entity-Relationship (ER) into Relational Model (RM) transformation. We have validated our proposal in a prototype developed under SWI-Prolog. The prototype together with the case study can be downloaded from `http://indalog.ual.es/mdd`.

The proposal is based on the use of logic programming with two ends. Firstly, specification of transformations. Secondly, specification of properties for model validation. Our approach adopts the OWL to logic programming mapping as basis. Firstly, transformations can be expressed in Prolog rules. In model transformation, a transformation maps the source model into the target model, and thus, the **ABox** of the source model into the **ABox** of the target model. Model transformation can be seen as a mapping of Prolog facts following the OWL to logic programming encoding, and it can be defined by Prolog rules. Secondly, model validation can be encoded with Prolog. Using Prolog atoms and some elements of Prolog meta programming we are able to validate source and target models as well as

transformations.

The advantages of the approach are the following. Firstly, the declarative nature of the specification, secondly, the use of a standardized language (Prolog), and the ability of executing transformations and automatically validate source and target models. Besides, the use of Prolog as validation language enriches the mechanisms of ODM and OWL constructors for expressing model requirements. While ODM is an OWL profile for UML models the expressivity power of ODM is limited, and model validation needs to express more complex requirements.

The structure of the paper is as follow. Section 2 will introduce the model transformation framework and will describe a case study of transformation. Section 3 will present the Prolog-based approach. Section 4 will show model validation. Section 5 will discuss related work. Finally, Section 6 will conclude and present future work.

2 Model Transformation

The elements to be considered in a ontology based transformation using Prolog as transformation language can be summarized as follows:

- We have to consider the *meta-model of the source model* which defines the elements occurring in the source model. Instances of the source meta-model are transformed by applying the transformation rules.
- We have to consider the *meta-model of the target model* which defines the elements occurring in the target model.
- We have to define *Prolog rules* for transforming instances of the source meta-model into instances of the target meta-model.
- We have to define *Prolog rules* for validating the transformation. The validation consists in source model validation, target model validation and transformation validation.

The question now is, how to express transformations and validations in Prolog? Our proposal is as follows.

- The ODM proposal provides a representation of UML models by an ontology. The **TBox** represents the meta-model and the **ABox** properly represents the model. We can represent the **ABox** by Prolog facts. Fortunately, SWI-Prolog, used in our prototype, is equipped with a library for RDF(S)/OWL which imports and exports RDF triples to Prolog facts.
- A transformation from a source model into a target model can be seen as a transformation from the set of Prolog facts of the source model into the set of Prolog facts representing the target model. Prolog rules can be used for transforming Prolog facts.
- Model validation consists in checking properties on source and target models in isolation as well as checking cross properties on both models. Model validation with Prolog consists in checking properties about the set of Prolog facts representing source and target models.

Our approach has been implemented and tested with some examples. We have used several UML and OWL tools (see Figure 1).

- We have used the *TopCased* UML tool [28] for designing the source and target meta-models.
- In addition, we made use of a UML2OWL transformer (available from [15]) in order to have the ODM-based representation of source and target meta-models.

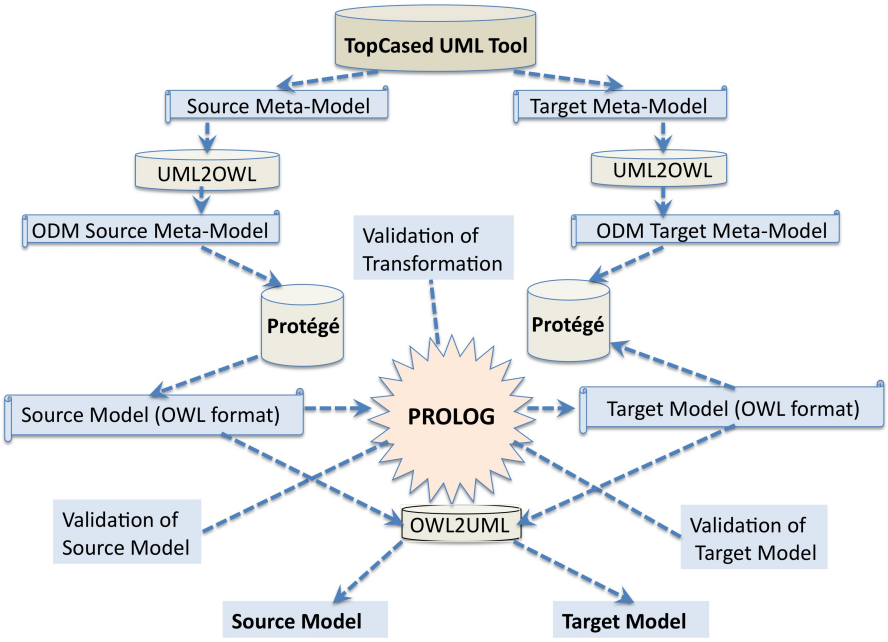

Figure 1: UML/OWL tools

– We have also employed the *Protégé* tool [18] for defining the instance of the source meta-model, and for exporting the source model (i.e. meta-model+instance) to an OWL document.
– After, the SWI-Prolog interpreter is utilized to validate the source model, and to transform the instance of the source model into the instance of the target model.
– Once the target model is computed, SWI-Prolog is used to validate the target model, and to validate the transformation.
– Next, the *Protégé* tool is also employed to export the target model together with the target meta-model to an OWL document.
– Finally, an OWL2UML transformer has been employed to obtain the target model from the ODM-based representation.

2.1 Case Study

In this section we will describe the case study of the paper. It is a well-known example of model transformation. Basically, the entity-relationship (ER) model is transformed into the relational model (RM). The model of Figure 2 represents the modeling of a database by an ER style diagram, while the model of Figure 3 is a RM style modeling of the same database.

The ER modeling of Figure 2 can be summarized as follows. *Data* are represented by classes (i.e., *Student* and *Course*), including attributes; *stores* are defined for each data (i.e., *DB_Students* and *DB_Courses*); *relations* are represented by associations; relation names are association names; besides, association ends are defined (i.e., *the_students*, *the_courses*, *is_registered* and *register*); relations can be

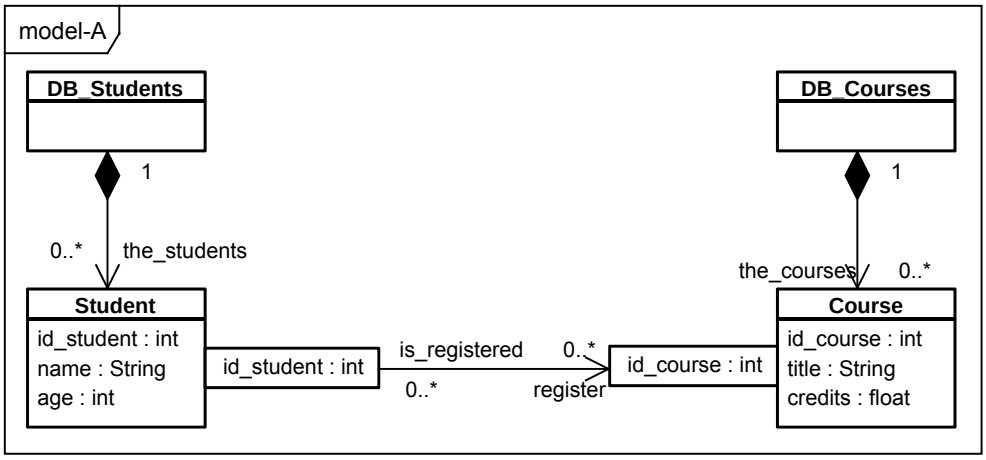

Figure 2: Entity-relationship modeling of the Case Study

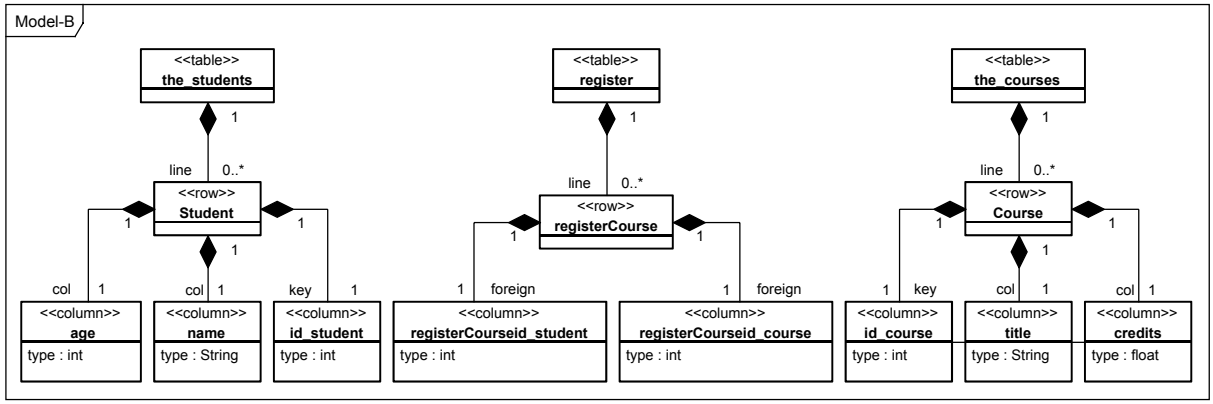

Figure 3: Relational modeling of the Case Study

adorned with *qualifiers* and navigability; qualifiers specify the key attributes of each data (used as foreign keys of the corresponding association).

Figure 3 shows the RM modeling of the same database. *Tables* are composed of *rows*, and rows are composed of *columns*. It introduces the following UML stereotypes: $<<$ *table* $>>$, $<<$ *row* $>>$ and $<<$ *column* $>>$. Furthermore, *line* is the role of the rows in the tables and *key*, *foreign* and *col* is the role of the key, foreign, and non key and non foreign attributes in rows, respectively. Finally, each column has an attribute *type*.

Figure 4 represents the meta-models of ER and RM models. In the first case, *DB_Students* and *DB_Courses* are instances of the class *store*, while *Student* and *Course* are instances of the class *data*, and the attributes of classes *Student* and *Course* are instances of the class *attribute*. In the second case, tables and rows of the target model are instances of the corresponding meta-classes, and the same can be said about classes *key*, *col* and *foreign*.

Now, the goal of the model transformation is to describe how to transform a class diagram of the type A (like Figure 2) into a class diagram of the type B (like Figure 3).

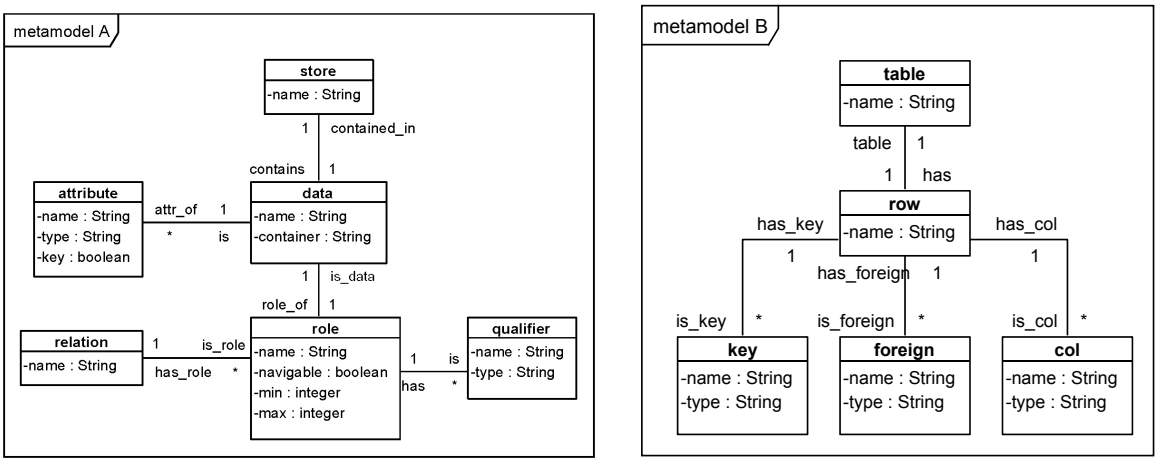

Figure 4: Meta-model of the Source/Target Models

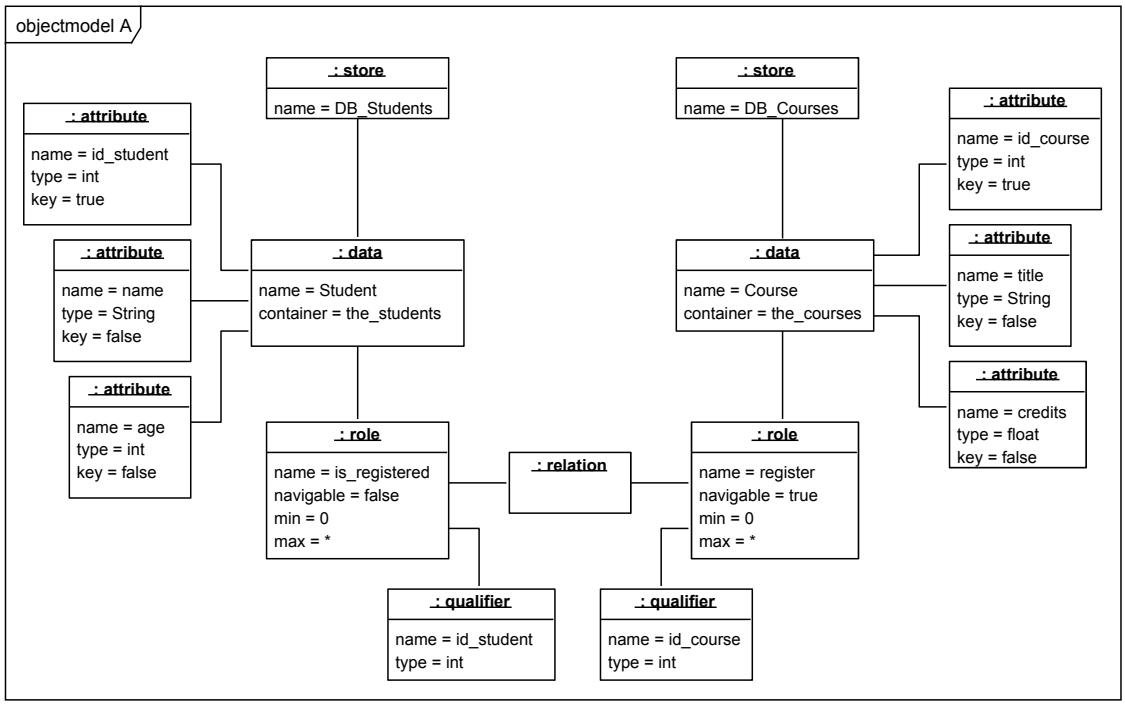

Figure 5: Object Model of Source Model

The transformation is as follows. The transformation generates two tables called *the_students* and *the_courses* each including three columns that are grouped into rows. The table *the_students* includes for each student the attributes of *Student* of Figure 2. The same can be said for the table *the_courses*. Given that the association between *Student* and *Course* is navigable from *Student* to *Course*, a table of pairs is generated to represent the assignments of students to courses, using the role name of the association end, that is, *register* concatenated with *Course*, that is, *registerCourse*, for naming the cited table. The

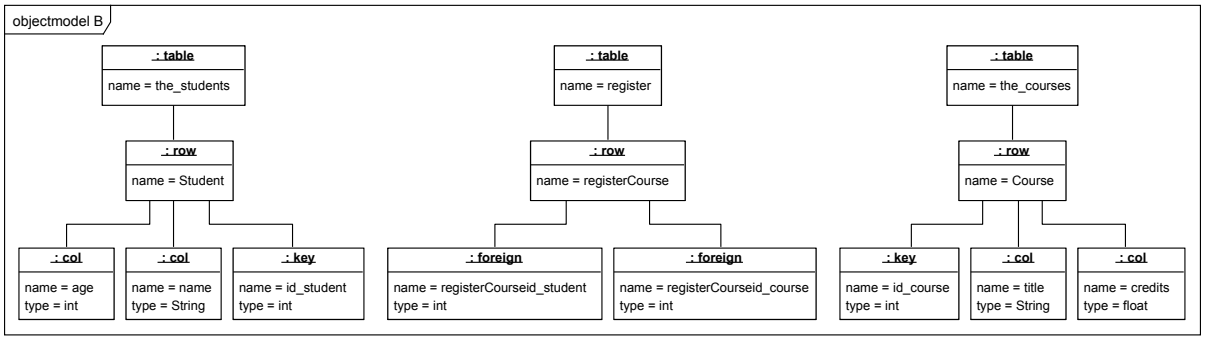

Figure 6: Object Model of Target Model

Source Model	
(1) All attributes of a data have distint names (SR) (WF)	(2) Each data has a unique key attribute (SR) (TR)
(3) Each data has a key attribute (SR) (TR)	(4) Each attribute is associated to exactly one data (SC) (WF)
(5) Each data is contained in exactly one store (SC) (WF)	(6) All data have distinct names (SR) (TR)
(7) All data have distinct containers (SR) (TR)	(8) Each qualifier is associated to exactly one role (SC) (TR)
(9) All qualifier names of a role are distinct (SR) (TR)	(10) All qualifiers are key attributes (SR) (WF)
(11) Each relation has two roles (SC) (WF)	(12) All relation names are distinct (SR) (WF)
(13) Each role is associated to exactly one relation (SC) (TR)	(14) Each role is associated to exactly one data (SC) (TR)
(15) All role names of a data are distinct (SR) (TR)	(16) Each store is associated to exactly one data (SC) (WF)
Target Model	
(17) All col names of a row are distinct (SR) (WF)	(18) All foreign names of a row are distinct (SR) (WF)
(19) All key names of a row are distinct (SR) (WF)	(20) All foreigns of a row are keys of another row (SR) (WF)
(21) Each table is associated to exactly one row (SC) (WF)	(22) Each row is associated to exactly one table (SC) (WF)
(23) Each key is associated to exactly one row (SC) (TR)	(24) Each col is associated to exactly one row (SC) (TR)
(25) Each foreign is associated to exactly one row (SC) (TR)	(26) All table names are distinct (SR) (WF)
(27) All row names are distinct (SR) (WF)	(28) All rows have exactly one key (SC) (TR)
(29) All rows have either all keys and cols or all foreigns (SR) (TR)	
Cross Requirements	
(30) Key and col names and types are names and types of attributes	(31) Table names are either container names or role names
(32) Row names are data names or concatenations of role and data names	(33) Foreign names are concatenations of roles, data and keys

Table 1: Model validation: requirements

columns *registerCourseid_student* and *registerCourseid_course* taken from qualifiers, play the role of foreign keys which are represented by the role *foreign* in the association of Figure 3.

The transformation can be considered as a transformation between object diagrams of source and target meta-models (see Figures 5 and 6). A transformation should be able to define a set of rules from which instances of the target meta-model are obtained from instances of the source meta-model.

2.2 Model Validation

While source and target meta-models impose *structural constraints (SC)* on *source* and *target models*, we can specify several *semantic requirements (SR)* on source and target models. In addition, we can describe cross requirements on models. In Table 1 we can see a set of requirements classified as (SC) and (SR). Some requirements express conditions on *well-formed models* (WF), while some of them are required by the transformation, that is, they are *transformation requirements* (TR).

– For instance, (2) requires that each data has a unique key attribute. This is a semantic requirement. Key attributes are attributes having *key* set to *true*, and the existence of a unique key attribute

cannot be expressed in the UML diagram. Moreover, this requirement is a pre-condition of the transformation because key attributes are used as foreign keys in the target model.

- Case (5) is an structural constraint on well-formed models: each data is associated to exactly one store. It is not needed in the transformation and can be expressed in the UML diagram with a cardinality constraint.
- Cases (6), (7), (9), (12) and (15) are related to naming of elements of source models, and therefore they are semantic requirements. They (except (12)) are required by the transformation: data and container names are used for naming tables and rows in the target model, while role and qualifier names (concatenated with data names) are also used for naming rows and foreign keys.
- (12) is required on a well-formed source model.
- In the target model tables, rows, cols, keys and foreigns are not shared (cases (21)-(25)).
- Case (20) is a semantic requirement that describes the relationship between foreign keys and keys in a well-formed target model.
- Case (29) is required by the transformation which assigns either keys and cols or foreigns to rows.
- Finally, cases (30)-(33) describe the relationship between names of the target model and names of the source model.

It is worth observing that the requirements about source and target models in isolation are not enough for the soundness of the transformation. For instance, source and target models can both have keys, but a cross requirement is needed: the keys of the target models are the keys of the source model.

3 Prolog for Model Transformation and Validation

In this section, we will show how Prolog can be used for defining transformation and validation rules in our approach. With this aim, we have to consider the following elements.

3.1 Prolog based Transformation

The Prolog interpreter has to import and export OWL files. This is the case of SWI-Prolog which includes a library to import and export RDF(S)/OWL triples. The SWI-Prolog library stores RDF triples in a database, and they can be retrieved with the predicate `rdf`. The RDF library includes predicates:

- `rdf_reset_db/0` which resets the database,
- `rdf_load(+File,+Options)` for importing triples,
- `rdf_save(+File)` for exporting triples,
- and finally, `rdf_assert(+Subject,+Property,+Object)` for inserting a new triple in the current database.

A Prolog predicate `transform(+SourceModelFile,+TargetModelFile)` has been defined to transform a source model (stored in a OWL file) into a target model (stored also in a OWL file). The Prolog code of such predicate is as follows:

```
transform(_,_):-rdf_reset_db,fail.
transform(_,_):-retractall(new(_,_,_)),fail.
transform(FileIn,_):-rdf_load(FileIn,[]),fail.
transform(_,_):-newrdf(A,B,C),assert(new(A,B,C)),fail.
transform(_,_):-rdf_reset_db,fail.
transform(_,_):-new(A,B,C),rdf_global_term(B,D),rdf_assert(A,D,C),fail.
transform(_,FileOut):-rdf_save(FileOut),rdf_reset_db.
```

The transformation rules define new triples representing the target model. Hence, a new predicate called `newrdf` is defined by the transformation rules. For instance, the following rules define the individuals of the class *table* of the model B from the model A of the case study:

```
newrdf(IdTable,rdf:type,'http://metamodelB.ecore#table'):-
          rdf(IdData,rdf:type,'http://metamodelA.ecore#data'),
          generate_id([IdData,'table1'],IdTable).

newrdf(IdTable,rdf:type,'http://metamodelB.ecore#table'):-
          rdf(IdRole,'http://metamodelA.ecore#role.navigable',E),
          E=literal(type(_,true)),
          generate_id([IdRole,'table2'],IdTable).
```

The first rule defines triples `(IdTable,rdf:type,'http://metamodelB.ecore#table')` obtained from triples `(IdData,rdf:type,'http://metamodelA.ecore#data')`, where `IdTable` is the identifier of the table, which is generated by the call `generate_id` from the data identifier `IdData` and `'table1'`.

The second rule defines the individuals of class *table* obtained from navigable roles, which are generated from the role identifier `IdRole` and `'table2'`.
In such a way that the following Prolog goal obtains the tables of the target model:

```
?- newrdf(IdTable,rdf:type,'http://metamodelB.ecore\#table').
    IdTable = 'http://metamodelA.ecore#02_Student_datatable1' ;
    IdTable = 'http://metamodelA.ecore#09_Course_datatable1' ;
    IdTable = 'http://metamodelA.ecore#13_register_roletable2' ;
false.
```

which represent the individuals of classes *Student*, *Course* and *register* of Figure 6. Now, the individuals of the class *row* of Figure 6 can be defined as follows:

```
newrdf(IdRow,rdf:type,'http://metamodelB.ecore#row'):-
          rdf(IdData,rdf:type,'http://metamodelA.ecore#data'),
          generate_id([IdData,'row1'],IdRow).

newrdf(IdRow,rdf:type,'http://metamodelB.ecore#row'):-
          rdf(IdData,'http://metamodelA.ecore#data.role_of',IdRole),
          rdf(IdRole,'http://metamodelA.ecore#role.navigable',E),
          E=literal(type(_,true)),
          generate_id([IdRole,IdData,'row2'],IdRow).
```

The first rule defines the individuals of the class *row* obtained from instances of data (i.e., the identifiers of *the_courses* and *the_students*), and the second rule defines the individuals of the class *row* obtained from navigable data roles (i.e., the identifier of *registerCourse*).

Now, *key*, *col* and *foreign* elements have to be defined. For instance, the individuals of the class *foreign* are defined as follows:

```
newrdf(IdForeign,rdf:type,'http://metamodelB.ecore#foreign'):-
            rdf(IdRole,'http://metamodelA.ecore#role.navigable',E),
            E=literal(type(_,true)),
            rdf(IdRole,'http://metamodelA.ecore#role.is',IdQualifier),
            rdf(IdData,'http://metamodelA.ecore#data.role_of',IdRole),
            generate_id([IdRole,IdData,IdQualifier,'foreign1'],IdForeign).


newrdf(IdForeign,rdf:type,'http://metamodelB.ecore#foreign'):-
            rdf(IdRole,'http://metamodelA.ecore#role.navigable',E),
            E=literal(type(_,true)),
            rdf(IdRole,'http://metamodelA.ecore#role.has_role',IdRelation),
            rdf(IdRelation,'http://metamodelA.ecore#relation.is_role',IdRole2),
            rdf(IdRole2,'http://metamodelA.ecore#role.is',IdQualifier),
            IdRole2\==IdRole,
            rdf(IdData,'http://metamodelA.ecore#data.role_of',IdRole),
            generate_id([IdRole2,IdData,IdQualifier,'foreign2'],N).
```

In this case, instances of the class *foreign* are obtained from navigable roles, using the identifier of the qualifier and the identifier of the role to generate the identifier. Now, the association roles of the Figure 5 have to be defined. For instance, the role *has* from the class *table* of Figure 4 is defined as follows:

```
newrdf(IdTable,'http://metamodelB.ecore#table.has',IdRow):-
      rdf(IdData,rdf:type,'http://metamodelA.ecore#data'),
      generate_id([IdData,'table1'],IdTable),
      generate_id([IdData,'row1'],IdRow).

newrdf(IdTable,'http://metamodelB.ecore#table.has',IdRow):-
        rdf(IdData,'http://metamodelA.ecore#data.role_of',IdRole),
        rdf(IdRolw,'http://metamodelA.ecore#role.navigable',E),
        E=literal(type(_,true)),
        generate_id([IdRole,'table2'],IdTable),
        generate_id([IdRole,IdData,'row2'],IdRow).
```

The first rule defines the rows of the tables obtained from instances of *data*, and the second rule defines the rows of the tables obtained from navigable roles.

Finally, attributes of the classes of the target metamodel of Figure 4 have to be defined. For instance, *name* of class *table* is defined as follows:

```
newrdf(IdTable,'http://metamodelB.ecore#table.name',Name):-
            rdf(IdData,'http://metamodelA.ecore#data.container',Name),
            generate_id([IdData,'table1'],IdTable).

newrdf(IdTable,'http://metamodelB.ecore#table.name',Name):-
          rdf(IdRole,'http://metamodelA.ecore#role.name',Name),
          rdf(IdRole,'http://metamodelA.ecore#role.navigable',C),
          C=literal(type(_,true)),
          generate_id([IdRole,'table2'],IdTable).
```

where the table names are obtained from container names (i.e., *the_students* and *the_courses*).

3.2 Prolog based Validation

Model validation is achieved with Prolog. Table 2 includes some of the Prolog rules of the requirements expressed in Table 1. The full set of rules can be downloaded from `http://indalog.ual.es/mdd`.

For validating the requirements on models, we can call the rules and in the case of success it indicates that the requirement is violated. In other words, the condition of the rule expresses the negation of

```
(1) attribute_distinct_names:-
              rdf(Data,mmA:'data.attr_of',Att1),rdf(Data,mmA:'data.attr_of',Att2),Att1\=Att2,
              rdf(Att1,mmA:'attribute.name',Name1),rdf(Att2,mmA:'attribute.name',Name2),
              Name1=Name2.
(2) exists_key:-
              setof(Att,(rdf(_,mmA:'data.attr_of',Att),
              rdf(Att,mmA:'attribute.key',literal(type(_,true)))),Keys),
              Keys=[].
(3) unique_key :-
              rdf(Data,mmA:'data.attr_of',Att1),rdf(Data,mmA:'data.attr_of',Att2),Att1\=Att2,
              rdf(Att1,mmA:'attribute.key',literal(type(_,true))),
              rdf(Att2,mmA:'attribute.key',literal(type(_,true))).
(5) unique_store_data:-
              setof(Store,rdf(_,mmA:'data.contained_in',Store),Stores),Stores=[_,_|_].
(10) qualifiers_are_keys :-
              rdf(_,mmA:'qualifier.name',Name),
              \+(rdf(Attribute,mmA:'attribute.name',Name),
                 rdf(Attribute,mmA:'attribute.key',literal(type(_,true)))).
(11) two_roles_relation :-
              setof(Role,rdf(_,mmA:'relation.is_role',Role),Roles),Roles\=[_,_].
(17) unique_col_names_row :-
              rdf(Row,mmB:'row.is_col',Col1),rdf(Row,mmB:'row.is_col',Col2),Col1\=Col2,
              rdf(Col1,mmB:'col:name',Name1),rdf(Col2,mmB:'col:name',Name2),Name1=Name2.
(20) foreign_keys:-
              rdf(Row,mmB:'row.is_foreign',Foreign),
              rdf(Row,mmB:'row.name',NRow),NRow=literal(type(_,RN)),
              rdf(Foreign,mmB:'foreign.name',NFor),NFor=literal(type(_,FN)),
              concat(RN,NKey,FN),
              \+rdf(_,mmB:'key.name',literal(type(_,NKey))).
(21) unique_table_row :-
              rdf(Table,mmB:'table.has',Row1),rdf(Table,mmB:'table.has',Row2),Row1\=Row2.
(29) well_formed_rows:-
              rdf(Row,mmB:'row.is_key',_),rdf(Row,mmB:'row.is_foreign',_).
     well_formed_rows:-
              rdf(Row,mmB:'row.is_col',_),rdf(Row,mmB:'row.is_foreign',_).
(31) containers_or_roles :-
              rdf(_,mmB:'table.name',Name),
              \+rdf(_,mmA:'role.name',Name),\+rdf(_,mmA:'data.container',Name).
(32) data_or_roles_and_data :-
              rdf(_,mmB:'row.name',Name),Name=literal(type(_,N)),
              \+rdf(_,mmA:'data.name',Name),
              \+(rdf(_,mmA:'role.name',Name1),rdf(_,mmA:'data.name',Name2),
                    Name1=literal(type(_,N1)),Name2=literal(type(_,N2)),concat(N1,N2,N)).
```

Table 2: Model validation: requirements

the requirement. Prolog meta programming predicates are used. For instance, case (2) uses the `setof` predicate to collect the set of keys of a given data.

4 Related Work

Validation and verification of model transformations is an emerging topic of research. We have found some similarities of our approach with the work proposed in [6]. The authors work in the context of the ATLAS Transformation language (ATL) and OCL, but handle the same kind of properties of our approach (unique names for relations and attributes together with existence of keys).

A more general framework for transformation validation and verification is proposed in [7] including verification and validation of properties about transformation rules. Our approach focused on properties

about meta-models, assuming that when some requirement is violated either source models or rules are incorrect.

Prolog has been also used in the *Model Manipulation Tool (MoMaT)* [27] for representing and verifying models. In [17] they propose consistency checking of class and sequence diagrams based on Prolog. Consistency checking rules as well as UML models are represented by Prolog, and a Prolog reasoning engine is used to automatically find inconsistencies.

On the other hand, logic programming based languages have already been explored in the context of model engineering in some works.

A first approach is [10], which describes the attempts to adopt several technologies for model transformation including logic programming. Particulary, they focused on *Mercury* and *F-Logic* logic languages. The approach [4] has introduced *inductive logic programming* in model transformation. The motivation of the work is that designers need to understand how to map source models into target models. With this aim, they are able to derive transformation rules from an initial and critical set of elements of the source and target models. The rules are generated in a (semi-) automatic way.

The *Tefkat* language [20, 19] is a declarative language whose syntax resembles a logic language with some differences (for instance, it incorporates a *forall* construct for traversing models). In this framework, in [13], they propose metamodel transformations in which evolutionary aspects are formalised using the Tefkat language.

In [11], they present a declarative approach for modeling requirements (designs and patterns) which are encoded as Prolog predicates. A search routine based on Prolog returns program fragments of the model implementation. Traceability and code generation are based on logic programming. They use *JTransformer*, which is a logic-based query and transformation engine for Java code, based on the Eclipse IDE.

Logic programming based model querying is studied in [9], in which logic-based facts represent meta-models. In [26] they study a transformation mechanism for the EMF Ecore platform using Prolog as rule-based engine. Prolog terms are used to represent models and predicates are used for deconstructing and reconstructing a term of a model.

Abductive logic programming is used in [14] for *reversible* model transformations, in which changes of the source model are computed from a given change of the target model. Finally, in [8], they have compared OCL and Prolog for querying UML models. They have found that Prolog is faster when execution time of queries is linear.

5 Conclusions and Future Work

In this paper we have presented a framework for the specification and validation of model transformations with Prolog rules, using the representation of UML models by ODM. Our approach has been applied to a well-known example of model transformation in which an UML class diagram representing a ER diagram is transformed into a UML diagram representing a relational database. We have validated our proposal with a prototype developed under SWI-Prolog.

Our approach has to be extended in the future as follows:

– Firstly, we would like to improve our prototype. Particularly, validation is now achieved by Prolog rules in which success and fail is returned. We would like to show more detailed analysis results, showing the model elements that violate the requirements, justifications, diagnosis, reparations, etc.

– Secondly, we would like to test our approach with other UML diagrams and transformations, and also with bigger examples;

– Thirdly, we are also interested in the use of our approach for model driven development of user interfaces in the line of our previous works [1, 2];

– Finally, we believe that our work will lead to the development of a logic based tool for transformation and validation of models.

References

[1] J. M. Almendros-Jiménez & L. Iribarne (2008): *An Extension of UML for the Modeling of WIMP User Interfaces*. Journal of Visual Languages and Computing, Elsevier 19, pp. 695–720, doi:10.1016/j.jvlc.2007.12.004.

[2] J. M. Almendros-Jiménez & L. Iribarne (2009): *UML Modeling of User and Database Interaction*. The Computer Journal 52(3), pp. 348–367, doi:10.1093/comjnl/bxn028.

[3] J.M. Almendros-Jiménez (2011): *A Prolog library for OWL RL*. In: Proceedings of the 4th International Workshop on Logic in Databases, ACM, pp. 49–56, doi:10.1145/1966357.1966367.

[4] Zoltán Balogh & Dániel Varró (2009): *Model Transformation by Example Using Inductive Logic Programming*. Software and Systems Modeling 8(3), pp. 347–364, doi:10.1007/s10270-008-0092-1.

[5] Alex Borgida (1996): *On the relative expressiveness of Description Logics and Predicate Logics*. Artificial Intelligence 82(1-2), pp. 353–367, doi:10.1016/0004-3702(96)00004-5.

[6] Fabian Büttner, Jordi Cabot & Martin Gogolla (2011): *On validation of ATL transformation rules by transformation models*. In: Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation, MoDeVVa, ACM, New York, NY, USA, pp. 9:1–9:8, doi:10.1145/2095654.2095666.

[7] J. Cabot, R. Claris, E. Guerra & J. de Lara (2010): *Verification and Validation of Declarative Model-to-Model Transformations*. Systems and Software 2(83), pp. 283–302, doi:10.1016/j.jss.2009.08.012.

[8] J. Chimia-Opoka, M. Felderer, C. Lenz & C. Lange (2008): *Querying UML models using OCL and Prolog: A performance study*. In: Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on, IEEE, pp. 81–88, doi:10.1109/ICSTW.2008.33.

[9] P. Dohrmann & S. Herold (2010): *Designing and Applying a Framework for Logic-Based Model Querying*. In: Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on, IEEE, pp. 164–171, doi:10.1109/SEAA.2010.21.

[10] A. Gerber, M. Lawley, K. Raymond, J. Steel & A. Wood (2002): *Transformation: The Missing Link of MDA*. In: Procs of ICGT'02, LNCS 2505, Springer, London, UK, pp. 90–105, doi:10.1007/3-540-45832-8_9.

[11] M. Goldberg & G. Wiener (2012): *A Declarative Approach for Software Modeling*. In: Practical Aspects of Declarative Languages, Springer LNCS 7149, pp. 18–32, doi:10.1007/978-3-642-27694-1_3.

[12] Benjamin N. Grosof, Ian Horrocks, Raphael Volz & Stefan Decker (2003): *Description Logic Programs: Combining Logic Programs with Description Logic*. In: Proc. of the International Conference on World Wide Web, ACM, USA, pp. 48–57, doi:10.1145/775152.775160.

[13] David I. Hearnden (2007): *Deltaware: Incremental Change Propagation for Automating Software Evolution in Model-Driven Architecture*. Ph.D. thesis, Centre or Institute School of Information Tech & Elec Engineering, Univ. of Queensland. Available at `http://espace.library.uq.edu.au/view/UQ:152739`.

[14] Thomas Hettel, Michael Lawley & Kerry Raymond (2009): *Towards Model Round-Trip Engineering: An Abductive Approach*. In: Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations, ICMT '09, Springer-Verlag, Berlin, Heidelberg, pp. 100–115, doi:10.1007/978-3-642-02408-5_8.

[15] Guillaume Hillairet (2007): *ATL Use Case - ODM Implementation (Bridging UML and OWL)*. Technical Report, http://www.eclipse.org/m2m/atl/atlTransformations/.

[16] Frédéric Jouault & Ivan Kurtev (2007): *On the interoperability of model-to-model transformation languages*. *Sci. Comput. Program.* 68(3), pp. 114–137, doi:10.1016/j.scico.2007.05.005.

[17] Zohaib Khai, Aamer Nadeem & Gang-soo Lee (2011): *A Prolog Based Approach to Consistency Checking of UML Class and Sequence Diagrams*. In Tai-hoon Kim, Hojjat Adeli, Haeng-kon Kim, Heau-jo Kang, Kyung Jung Kim, Akingbehin Kiumi & Byeong-Ho Kang, editors: *Software Engineering, Business Continuity, and Education*, *Communications in Computer and Information Science* 257, Springer Berlin Heidelberg, pp. 85–96, doi:10.1007/978-3-642-27207-3_10.

[18] H. Knublauch, R.W. Fergerson, N.F. Noy & M.A. Musen (2004): *The Protege OWL Plugin: An Open Development Environment for Semantic Web Applications*. In: *The Semantic Web, ISWC 2004: Third International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004: proceedings*, LNCS 3298, Springer-Verlag New York Inc, pp. 229–243, doi:10.1007/978-3-540-30475-3_17.

[19] M. Lawley & J. Steel (2006): *Practical Declarative Model Transformation with Tefkat*. In: *MoDELS Satellite Events*, LNCS 3844, Springer, pp. 139–150, doi:10.1007/11663430_15.

[20] Michael Lawley & Kerry Raymond (2007): *Implementing a practical declarative logic-based model transformation engine*. In: *SAC'07: Proceedings of the 2007 ACM Symposium on Applied Computing*, ACM, New York, NY, USA, pp. 971–977, doi:10.1145/1244002.1244216.

[21] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue & Carsten Lutz (2009): *OWL 2 Web Ontology: Reasoning in OWL 2 RL and RDF Graphs using Rules*. Technical Report, http://www.w3.org.

[22] OMG (2003): *MDA Spec.* Technical Report, http://www.omg.org/mda/specs.htm.

[23] OMG (2008): *Model Driven Architecture*. Technical Report, http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[24] OMG (2008): *Object Constraint Language (OCL)*. Technical Report, http://www.omg.org/technology/documents/formal/ocl.htm.

[25] OMG (2009): *Ontology Definition Metamodel (ODM)* . Technical Report, http://www.omg.org/spec/ODM/1.0/.

[26] Bernhard Schätz (2009): *Formalization and Rule-Based Transformation of EMF Ecore-Based Models*. In Dragan Gašević, Ralf Lämmel & Eric Wyk, editors: *Software Language Engineering*, Springer-Verlag, Berlin, Heidelberg, pp. 227–244, doi:10.1007/978-3-642-00434-6_15.

[27] H. Storrle (2007): *A Prolog-based Approach to Representing and Querying UML Models*. In: *Intl. Ws. Visual Languages and Logic (VLL'07)*, 274, pp. 71–84. Available at http://ceur-ws.org/Vol-274/.

[28] TopCased (2010): *TopCased: The Open-source tool kit for critical systems*. Technical Report, http://www.topcased.org/.

[29] Laurence Tratt (2005): *Model transformations and tool integration*. *Software and System Modeling* 4(2), pp. 112–122, doi:10.1007/s10270-004-0070-1.