Transformation of Binary Relationships with Particular Multiplicity*

Zdeněk Rybola[1] and Karel Richta[2]

[1]Department of Software Engineering, Faculty of Information Technology, Czech Technical University in Prague
[2]Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague
[2]Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague
richta@ksi.mff.cuni.cz

Abstract. The paper deals with one small step in the process of model driven development (MDD) or model driven architecture (MDA) – widely used terms nowadays. MDD defines techniques to develop software systems using variety of models together with a set of transformations. MDD specifies several levels of models depending on abstraction – ranging from computation independent models (CIM) to platform independent models (PIM) into platform specific models (PSM), and implementation specific models (ISM). Many CASE tools provide automated support for generating more specific models from more abstract ones – e.g. PSM or ISM from PIM. This task is referred to as forward engineering. Many tools also provide automated support for generating more abstract models from more specific ones – e.g. PIM from PSM or ISM. This task is referred to as reverse engineering. Some tools also support round-trip engineering, which involves both forward and reverse engineering steps such that software artifacts (model and code) become synchronized. However, these tools need exact transformation rules to be defined for such transformations. This paper describes basic principles and restrictions for transformations of binary relationships and transformations of binary relationships with the particular multiplicity from PIM level into PSM level. The idea is illustrated on examples.

1 Introduction

Model driven development is the dream of OMG (Object Management Group), and the top desire is so called round-trip engineering. It is the combination of a forward process (generating code from model) and a reverse process (generating model from code). The necessary equipment for such round-trip process are transformations, which serve as the description of partial steps in this process. OMG offers Unified Modeling Language (UML) for the description of models.

* This work has been partially supported by the grant project of the Czech Grant Agency (GAČR) No. GA201/09/0990 and by SGS grant.

The necessary part of such notation is a language for description of model constraints – in the case of UML it is Object Constraint Language (OCL) – one of the basic parts of UML.

OMG defines MDA or MDD [6] as the set of modeling levels or views. These models can be transformed – from the upper level to a lower one, or from the lower level to an upper one, or between models on the same level of abstraction – some sort of model refactoring. Transformations have to be described precisely in some formal fashion.

OCL [3,10,7] is a specification language. As a part of UML standard [5,4], it is used to define restrictions for the model in UML by constructs such as invariants, pre- and post-conditions connected to model elements which give context for the constraints.

This paper deals with the basic principles of transformations of binary relationships in PIM level into PSM level. To illustrate these principles we suppose the relational database system as the platform, so the SQL serves as the platform specific language. Where particular restrictions are required for transformation of binary relationship to PSM, OCL constraints in PIM level are defined first, so these can be used in transformations to other PSMs or by automated transformation tools such as [9,2,13]. These tools provide support to OCL syntax checking, constraints evaluation for system snapshots and even generating of source code for connected model and constraints.

The paper is structured as follows. Section 3 defines the binary relationship and its multiplicities. Transformation techniques for relationships with common multiplicity values is presented in section 4. Particular multiplicities and their transformation is explained in section 5. Example of full PIM transformation with relationships with particular multiplicity values is shown in section 6 and final conclusions are given in section 7.

2 Existing tools for model and constraint transformation

There are several tools that provide support for model and OCL constraint evaluation and transformation.

Enterprise Architect [12] is a CASE tools for creating and managing models. It supports model transformation, source code generation and reverse engineering from source code to PSM models. For instance, it includes transformation from platform independent class model to platform specific database model and also generation of SQL source code from the database model. However, these transformations does not consider the minimal multiplicities of a relationship for the required constraints as described further in this paper.

Dresden OCL Toolkit [2,13] is a Eclipse plugin. The toolkit can load a UML class model and OCL constraints. It can load a model instance and evaluate the constraints for it. It can also generate SQL code for a database to create and constraints to check in it or AspecJ source code for constraints checking in Java. However, even this toolkit does not consider the minimal multiplicities of a relationship for the foreign key direction, nullability and uniqueness or specific OCL constraints creation to ensure the multiplicity.

Therefore we want to define the multiplicity constraints in a formal way – in OCL – so they can be adapted in such a tool.

3 Binary relationship and its multiplicities

The conceptual model uses entities or classes as the representation of the types of objects, and associations between them representing relationships between entities. Relationships can be unary (properties of objects), binary (an association between two objects), or n-ary (an association between n objects). It can be proved, that n-ary relationships can be substituted by (n-1) binary relationships without loss of generality [11]. Therefore, we can elaborate only the binary relationships.

Binary relationship is a connection between two entities. It means instances of one entity has a relationship to an instance of the other one (however, it can be the same entity as well).

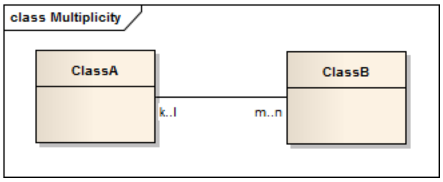

Fig. 1. Labeling of minimal and maximal multiplicities of a relationship in UML class diagram

There are several types of relationship depending on multiplicities of the relationship. The multiplicity defines the number of instances connected to each other. Fig. 1 shows general form of modeling binary relationships using UML class diagram notation [4,5,1]. Multiplicities of the relationship are labeled by parameters k, l, m and n with the following meaning:

- k represents minimal number of instances of ClassA related to an instance of ClassB
- l represents maximal number of instances of ClassA related to an instance of ClassB
- m represents minimal number of instances of ClassB related to an instance of ClassA
- n represents maximal number of instances of ClassB related to an instance of ClassA.

The minimal multiplicity of a relationship indicates the obligation of an instance to be related to instance or instances of the other type. Minimal multiplicity equal to *zero* means that there is no obligation for the source instance to be related to any instance of the target type. On the other hand, when the minimal multiplicity is equal to *one*, it means that the source instance must be related to at least one target instance. The former is actually no restriction at all. The latter is a restriction and it can be expressed using OCL constraint as follows:

```
context a:ClassA inv minBdirect:
not a.b.asSet()->isEmpty()
```

Because the relationship can be unidirectional – and it usually is, for instance in PSM for relational database using foreign keys – it is useful in such case to define the constraint in reverse direction like the following:

```
context a:ClassA inv minBreverse:
ClassB.allInstances()->exists(b | b.a = a).
```

The maximal multiplicity of relationship indicates if the instance can be related just to a single target instance or to a set of target instances. The maximal multiplicity of *one* stands for an instance of source type being related to a single instance of target type, while *asterisk (*)* stands for a set of instances of target type related to a single instance of the source type. The latter one is actually no restriction while the former is a restriction and it can be expressed using OCL constraint as follows:

```
context a:ClassA inv maxBdirect:
a.b.asSet()->size() <= 1
```

or with perspective of unidirectional relationship as follows:

```
context a:ClassA inv maxBreverse:
ClassB.allInstances()->count(b|b.a=a) <= 1
```

However, minimal and maximal multiplicities can take other values. We call such multiplicities particular ones and deal with them in section 5.

4 Transformation of binary relationships to PSM of relational database

In relational databases entities are represented by tables of rows, while rows represent instances of entities [11,8]. Each row can be identified by a mechanism called primary key, which ensures that each record is identified by the nonempty unique key. Relationships between instances of entities are represented by a mechanism of foreign keys, which links a row in the table (representing source entity) to a single row in a table representing target entity – using the target's primary key.

The mechanism of foreign and primary keys brings two main problems. Using this mechanism, any single source row can refer just to a single target row, not a set of rows, and therefore it can only realize one-to-one or one-to-many relationship. This is in contrast to conceptual modeling on PIM level where many-to-many relationships are also possible (and very common). Second problem is that the connection represented by foreign key is unidirectional in contrast to bidirectional relationships used in PIMs. That means in PSM of relational database, source entity has direct link to target entity while target entity does not have direct link back.

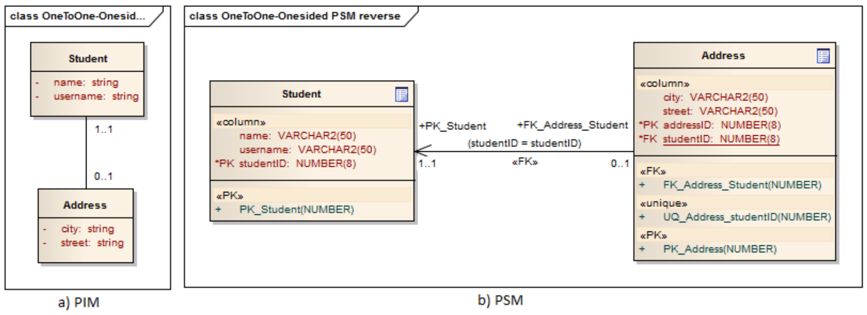

Fig. 2. a) PIM and b) PSM of one-to-one relationship with one side required

However, according to Fig. 2 where the one-to-one relationship is realized by foreign key referring from the table Address to the table Student, this mechanism automatically ensures that the maximal multiplicity of target entity in relationship is always equal to one (l=1). Furthermore, we can determine the minimal target multiplicity to zero (k=0) by making the foreign key *nullable* (i.e. it can be null, referring no target table row) or to one (k=1) by making it *not nullable* (i.e. must not be null, must refer some target table row). Because the foreign key is unidirectional, there is no restriction to the maximal multiplicity of the source entity in relationship and it is implicitly infinite. However, we can restrict it to one (n=1) by making the foreign key *unique*.

There is no way to restrict the minimal multiplicity of the source entity in the relationship (multiplicity m in Fig. 1) using just the foreign key. [11] suggests using *on delete restrict* clause for the foreign key. However, this clause only restricts deletion but not existence of related instance in general. Therefore we suggest defining a special constraint to restrict it to one (m=1) if required. This constraint is defined in sections 4.1 and 4.2. There are also some other restrictions for the realization by the foreign key mechanism depending on the multiplicity of relationship described in sections 4.1, 4.2 and 4.3.

4.1 Transformation of one-to-one relationship

In one-to-one relationship the direction of the foreign key differs depending on the minimal multiplicities of the relationship.

If both minimal multiplicities of the relationship are equal to zero, the direction of the foreign key does not matter in this case.

If the minimal multiplicity of one side of the relationship is equal to one, the direction of the foreign key realization is given as follows: the foreign key must be *not nullable* and must refer from the table representing the not required entity referring to a primary key in the table representing the required entity as shown in Fig. 2b.

If both minimal multiplicities of the relationship are equal to one, the relationship is required by both sides. In this situation there are two possible methods to realize the required relationship in PSM for relational database: using single foreign key; or using pair of reverse foreign keys.

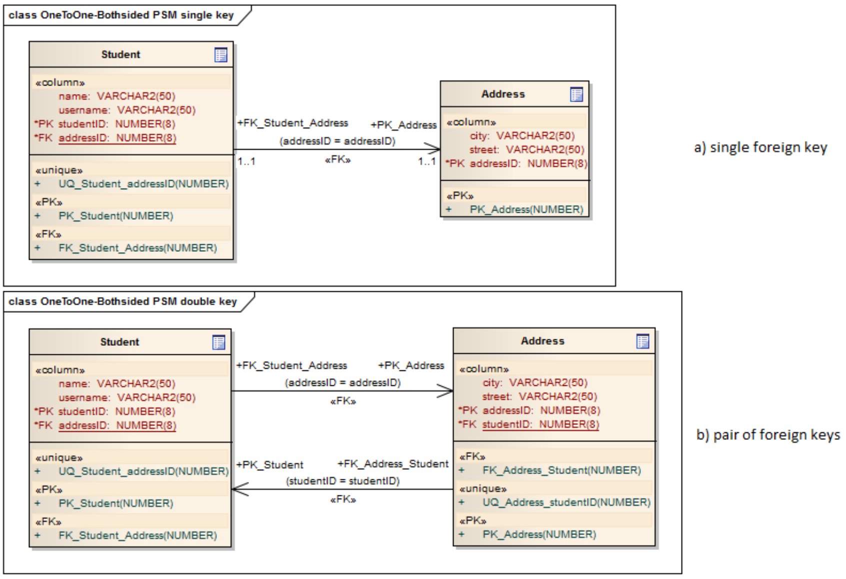

Fig. 3. PSM of one-to-one both-side required relationship with a) single foreign key and b) pair of foreign keys

Using single foreign key. In this case, the direction of the foreign key does not matter again like in the situation where both minimal multiplicities are zero. Let's consider the realization shown in Fig. 3a. To ensure minimal multiplicity k=1 while using only a single foreign key the realization of the reverse definition of the constraint for minimal multiplicity defined in section 3 is necessary to check for existence of the other related instance. The OCL constraint can be realized in SQL as view selecting records violating the multiplicity restrictions. The constraint view for the situation in Fig. 3 can be defined as follows:

```
CREATE VIEW violating_address AS
SELECT a.addressID FROM Address a
WHERE NOT EXISTS (SELECT 1 FROM Student s WHERE s.addressID = a.addressID);
```

This constraint can be also used to realize one-to-one relationship with one side required using the foreign key of reverse direction, i.e. the foreign key referring from the table of the required entity to a primary key in the table of the not required entity. In this case this constraint must be defined to ensure required minimal multiplicity in the opposite direction k=1 and the foreign key is *nullable* to enable minimal multiplicity m=0.

Using pair of reverse foreign keys. Another possible solution to ensure both minimal multiplicities k=1 and m=1 in one-to-one both-side required relationship is using pair of reverse *not null unique* foreign keys as shown in Fig. 3b. However, additional constraint must be defined to make sure both foreign keys refer the same instances, i.e. there is no circle of relations between a set of instances of both types like *s1 -> a1 -> s2 -> a2 -> s1*. The constraint can be defined in PIM in OCL as follows:

```
context s:Student inv notCircular: s.add.std = s
```

In PSM for relational database the constraint can be realized as view selecting records violating this constraint as follows:

```
CREATE VIEW violating_circular_students AS
SELECT s.studentID FROM Student s, Address a
WHERE s.addressID = a.addressID and a.studentID <> s.studentID;
```

Single table realization of one-to-one relationship. One-to-one relationship in general can be also realized by a single table representing both related entities. Such table contains all attributes of both entities and each row in such table stands for the whole relationship of instances while in case of no instance related the attributes of not participating instance stay *null*.

However, this realization violates third normal form [11] and the intention of designer to represent the entities separately. If the designer intends to realize the relationship in a single table then he can make this transformation in PIM already.

We prefer the realization using two separate tables and relationship realized by single foreign keys with additional constraints as described above.

4.2 Transformation of one-to-many relationship

One-to-many relationship is realized in PSM for relational database using foreign key like in the realization of one-to-one relationship. In this situation the direction of the key is strictly defined by the maximal multiplicities – the foreign

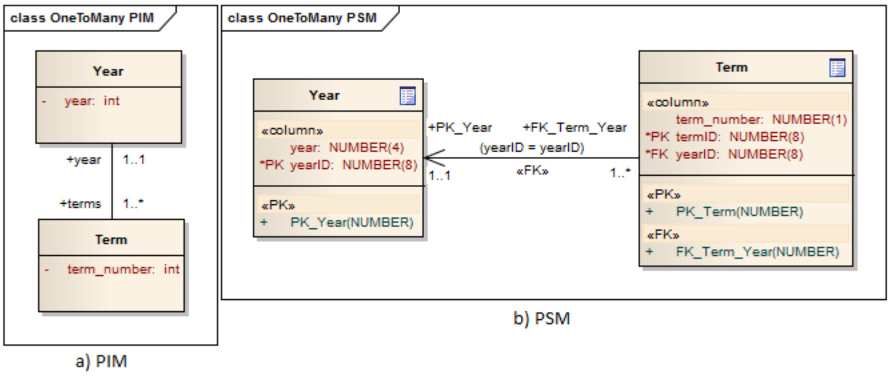

Fig. 4. a) PIM and b) PSM of one-to-many relationship

key must refer from the table representing the entity with higher multiplicity (source) to the table representing the entity with maximal multiplicity of one (target). Transformed PSM for the situation in Fig. 4a is shown in Fig. 4b.

Maximal multiplicity l=1 is ensured by the foreign key mechanism. To enable maximal multiplicity of the other side n=* the foreign key is *not unique*, so many source rows can refer the same target row. If the target entity is required in the relationship (i.e. the minimal multiplicity of target is k=1), the foreign key must be *not null* to ensure the requirement of being related to at least one target record. Finally, if the source entity type is required in the relationship (i.e. the minimal multiplicity of source entity is m=1), additional constraint must be defined and realized as shown in section 4.1.

The constraint can be realized in PSM for relational database by the following view selecting records that violate the required multiplicity:

```
CREATE VIEW violating_years AS
SELECT y.yearID FROM Year y
WHERE NOT EXISTS (SELECT 1 FROM Term t WHERE t.yearID = y.yearID);
```

4.3 Transformation of many-to-many relationship

It has been said before that relationships in relational databases are realized by using foreign key which is unidirectional and always refers only to a single row in a table. To realize many-to-many relationship we need to decompose the relationship into two one-to-many relationships and an entity to represent the original relationship (association entity) [11]. Minimal and maximal multiplicities of the original relationship become minimal and maximal multiplicities of the association entity in the appropriate one-to-many relationship. Minimal and maximal multiplicity of the original related entities become one in the relationship to the association entity.

Example of the decomposition of the relationship in PIM level is shown in Fig. 5a and 5b. The many-to-many relationship of entities Class and Registration

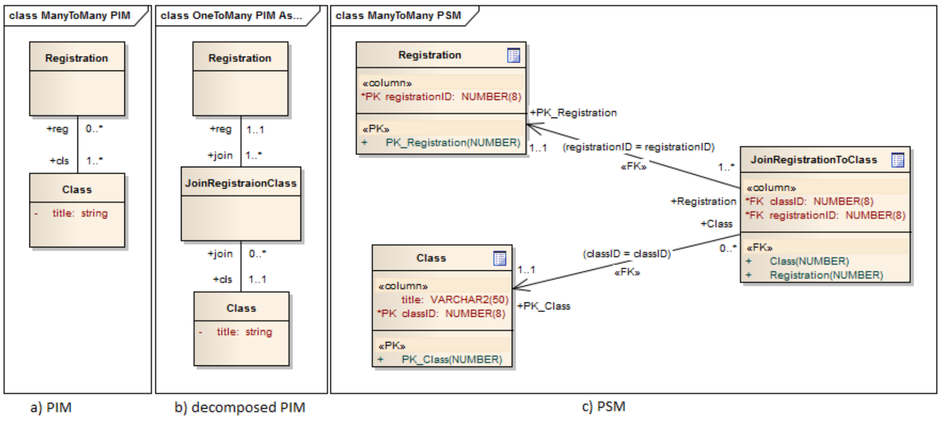

Fig. 5. PIM of many-to-many relationship

has been decomposed to an entity JoinRegistrationClass representing the relationship between a class and a registration and two one-to-many relationships of Class and JoinRegistrationClass and Registration and JoinRegistrationClass.

After this decomposition the transformation for both one-to-many relationships can be applied as shown in section 4.2. The resulting realization of the many-to-many relationship is shown in Fig. 5c.

5 Binary relationship with particular multiplicity

We have presented methods to transform typical cases of binary relationships from PIM to PSM for relational database. We have defined constraints for typical minimal and maximal multiplicities of relationships and their transformation to SQL views to select rows violating multiplicity constraints.

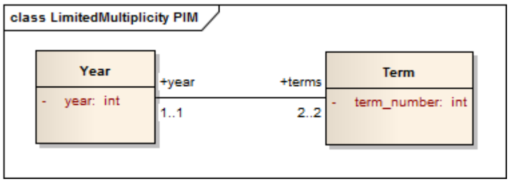

Fig. 6. PIM of one-to-many relationship with particular multiplicity

In many practical modeling situations the multiplicities of relationships in PIM are not restricted to those typical situations mentioned before. The multiplicities can be more restrictive to define exact number of instances being related together. We use the term *particular multiplicity* for minimal multiplicity greater than *one* or for maximal multiplicity other than *** or *1*. An example of such situation of strongly restricted multiplicities is shown in Fig. 6. The figure presents the situation of years and terms again, but this time a year consists of exactly two terms.

The type of relationship and its transformation is still defined by the maximal multiplicities as shown in Section 4. To restrict the particular multiplicity, additional constraints must be defined for minimal and maximal multiplicities. Using the notation shown in Fig. 1 with perspective of unidirectional realization of relationships these constraints can be defined in OCL as follows:

```
context a:ClassA inv minA:
ClassB.allInstances()->count(b | b.a = a) >= m
```

for the minimal multiplicity restriction and

```
context a:ClassA inv minA:
ClassB.allInstances()->count(b | b.a = a) <= n
```

for the maximal multiplicity restriction. If both minimal and maximal multiplicity of one side of a relationship are restricted to particular values, both constraints can be joined together into single constraint:

```
context a:ClassA inv minA:
ClassB.allInstances()->count(b | b.a = a) >= m
&
ClassB.allInstances()->count(b | b.a = a) <= n
```

5.1 Transformation of particular multiplicity constraints

Relationships with particular multiplicities are transformed to PSM for relational database the same way as binary relationships with typical multiplicities as defined and shown in section 4. Additionally, the defined constraints must be realized in SQL to ensure required multiplicity values. This can be done for situation from Fig. 6 by defining views selecting records violating the restricted multiplicities as follows:

```
CREATE VIEW violating_years_minimal AS
SELECT y.yearID FROM Year y
WHERE 2 > (SELECT count(*) FROM Term t WHERE t.yearID = y.yearID);
```

for the minimal multiplicity m=2 and

```
CREATE VIEW violating_years_maximal AS
SELECT y.yearID FROM Year y
WHERE 2 < (SELECT count(*) FROM Term t WHERE t.yearID = y.yearID);
```

for the maximal multiplicity n=2. In this situation, both minimal and maximal multiplicities are restricted to particular values, so a single view can be defined selecting records of years violating the restricted multiplicities generally:

```
CREATE VIEW violating_years AS
SELECT y.yearID FROM Year y
WHERE 2 <> (SELECT count(*) FROM Term t WHERE t.yearID = y.yearID);
```

6 Example transformation

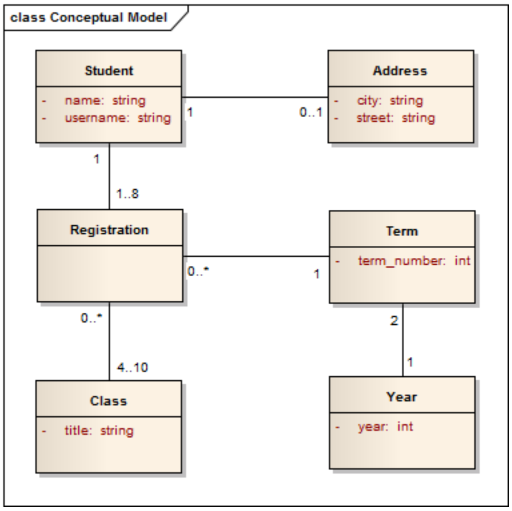

Fig. 7. PIM of the example

To illustrate binary relationships between entities and their transformations to PSM for relational database we prepared an example of a small school system for evidence of students and their registrations of classes to set of terms. Students make a registration of four to ten classes each term they study while each school year consist of exactly two terms. Each student can make eight registrations at most but, on the other hand, it must be registered at least into one term to appear in the system. Students may have address assigned but there might be no addresses without a student assigned. Fig. 7 presents UML class diagram[1] for the PIM of the example.

We chose transformation of PIM to PSM using the single foreign key to represent binary relationships between entities with additional constraints defined in OCL a realized in SQL as views as shown in this paper. The PSM is shown in Fig. 8 and the constraints defined as follows:

```
context s:Student inv countRegistrations:
Registration.allInstances()->count(r | r.std = s) >= 1
&
Registration.allInstances()->count(r | r.std = s) <= 8
```

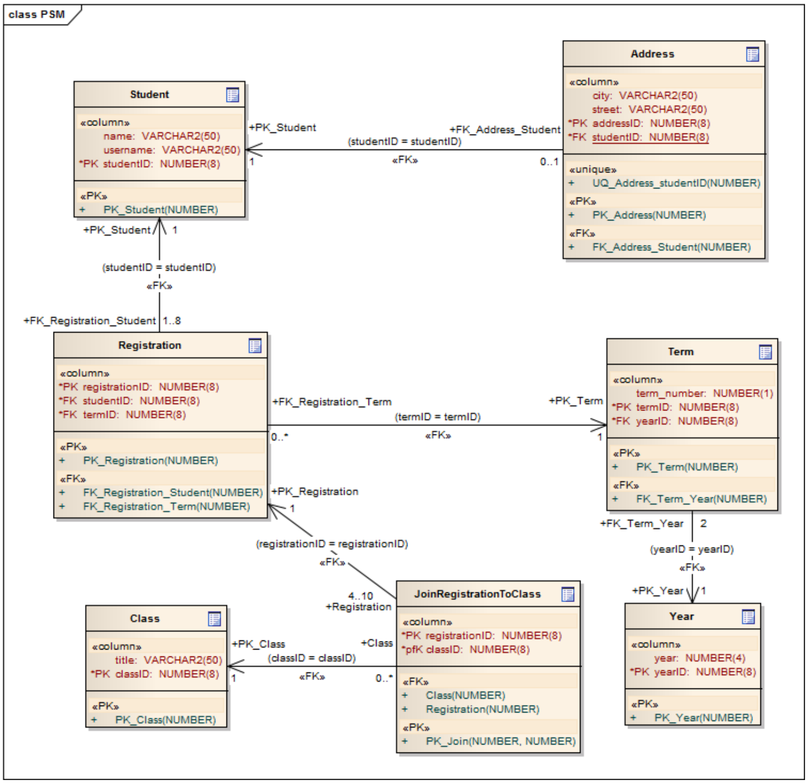

Fig. 8. Transformed PSM of the example

for multiplicity from *1* to *8* of relationship between Student and Registration,

```
context y:Year inv countTerms:
Term.allInstances()->count(t | t.year = y) = 2
```

for multiplicity *2* of relationship between Year and Term, and

```
context r:Registration inv countClasses:
Class.allInstances()->count(c | c.reg = r) >= 4
&
Class.allInstances()->count(c | c.reg = r) <= 10
```

for multiplicity from *4* to *10* of relationship between Registration and Class.

In PSM the defined constraints are realized by views selecting records violating multiplicity restrictions as follows:

```
CREATE VIEW violating_countRegistrations AS
SELECT s.studentID FROM Student s
WHERE (SELECT count(*) FROM Registration r
       WHERE r.studentID = s.studentID) NOT BETWEEN (1,8);
```

for multiplicity from *1* to *8* of relationship between Student and Registration,

```
CREATE VIEW violating_countTerms AS
SELECT y.yearID FROM Year y
WHERE (SELECT count(*) FROM Term t WHERE t.yearID = y.yearID) <> 2;
```

for multiplicity *2* of relationship between Year and Term, and

```
CREATE VIEW violating_countClasses AS
SELECT r.registrationID FROM Registration r
WHERE (SELECT count(*) FROM JoinRegistrationToClasses j
       WHERE j.registrationID = r.registrationID) NOT BETWEEN (4,10);
```

7 Conclusions

In this paper we presented transformations of binary relationships from PIM to PSM for relational databases. We presented basic principles of transformations of relationships according to their multiplicities to relational tables with the mechanism of foreign and primary keys. We also suggested restrictions associated with the relationship realization in PSM to ensure required minimal multiplicities in relationships. We defined these restrictions in PIM using OCL constraints so automated tools can use them for their transformations. We also presented examples of transformation of such constraints to PSM for relational databases as views selecting rows that violate the required multiplicities.

We also defined particular multiplicities of relationships and their expression using OCL constraints. We suggested its transformations to PSM for relational database and showed examples for illustration. We also presented a full example of PIM transformation to PSM for relational database with required constraints as suggested and defined in this paper.

References

1. Arlow, J., Neustadt, I.: UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition). Addison-Wesley Professional (2005)
2. Demuth, B.: DresdenOCL. http://www.reuseware.org/index.php/DresdenOCL (Jan 2011)
3. OMG: Object constraint language, version 1.3. http://www.omg.org/spec/OCL/2.2/PDF (Feb 2010)
4. OMG: Object management group - UML. http://www.uml.org (Feb 2011)
5. OMG: UML 2.3. http://www.omg.org/spec/UML/2.3/ (Feb 2011)
6. OMG, Miller, J., Mukerji, J.: MDA guide version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf (Jun 2003)
7. Richta, K.: Jazyk OCL a modelem řízený vývoj. In: Moderní databáze - Architektura moderních IS 2010 (2010)
8. Richta, K.: Rekonstrukce OCL z SQL. In: Datakon 2010 (2010)
9. Richters, M., Büttner, F., Gutsche, F., Kuhlmann, M.: USE - a UML-based specification environment. http://www.db.informatik.uni-bremen.de/projects/USE/ (Jan 2011)
10. Richters, M., Gogolla, M.: OCL: syntax, semantics, and tools. In: Object Modeling with the OCL, The Rationale behind the Object Constraint Language. pp. 42–68. Springer-Verlag (2002)
11. Rob, P., Coronel, C.: Database Systems: Design, Implementation, and Management. Boyd & Fraser, 2nd edn. (1995)
12. Sparx Systems: Enterprise architect - UML design tools and UML CASE tools for software development. http://www.sparxsystems.com.au/products/ea/index.html (Mar 2011)
13. Wilke, C., Thiele, M., Freitag, B.: Dresden OCL: manual for installation, use and development (Oct 2010)