On Static and Dynamic Analysis of UML and OCL Transformation Models

Martin Gogolla, Lars Hamann, Frank Hilken

Database Systems Group, University of Bremen, Germany
{gogolla|lhamann|fhilken}@informatik.uni-bremen.de

Abstract. This contribution discusses model transformations in the form of transformation models that connect a source and a target metamodel. The transformation model is statically analyzed within a UML and OCL tool by giving each constraint an individual representation in the underlying class diagram by highlighting the employed model elements. We also discuss how to analyze transformation models dynamically on the basis of a model validator translating UML and OCL into relational logic. One can specify, for example, the transformation source and let the tool compute automatically the transformation target on the basis of the transformation model without the need for implementing the transformation. Properties like injectivity of the transformation can be checked through the construction of example transformation pairs.
Keywords. Transformation model, Metamodel, UML, OCL, Model validator, Static and dynamic transformation model analysis.

1 Introduction

Model transformations are regarded as essential cornerstones for Model-Driven Engineering (MDE). Quality assessment and improvement techniques like transformation validation and verification are thus central for the success of MDE. Therefore, testing and analysis techniques for model transformations [2, 1] are obtaining more and more attention.

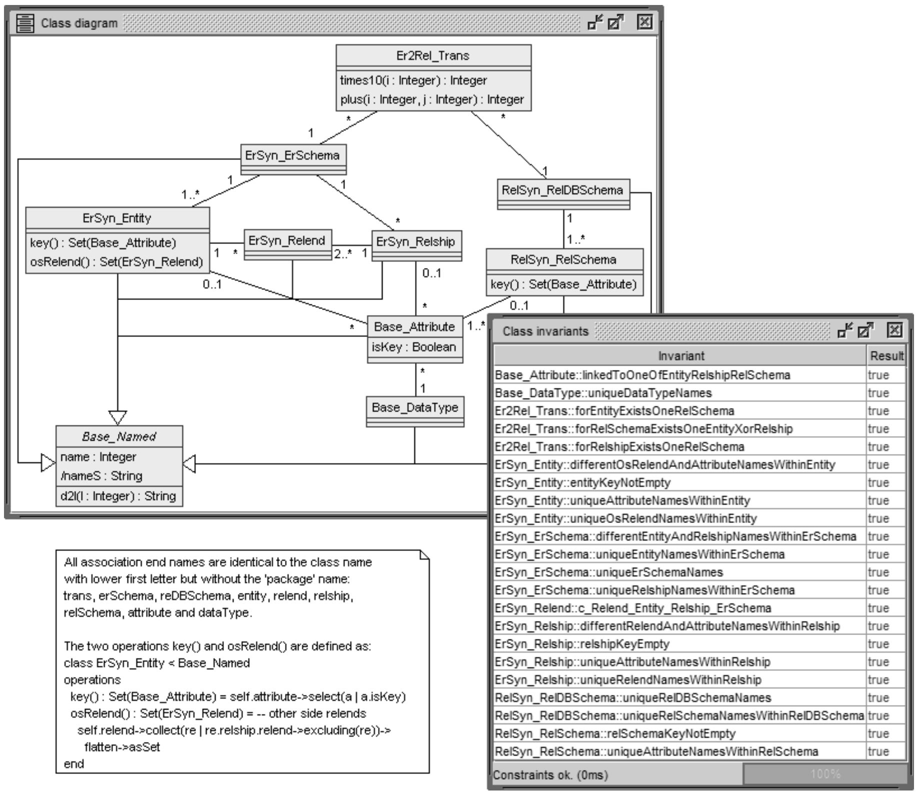
Here, we discuss model transformations in form of transformation models [3]. Transformation models are descriptive characterizations of mappings between a source and target metamodel. Our approach proposes to check the covering of constraints within transformation models statically in order to better understand the model, and to check for the completion of partially specified transformation pairs. We apply a so-called model validator in the tool USE (Uml-based Specification Environment) that searches for instances within a finite search space.

Our work has links to related approaches. Our contribution is based on Alloy [8] and Kodkod [10]. The implementation of the model validator that we employ is grounded on a translation of UML and OCL concepts into relational logic as described in [9]. Transformation models using the same example as here, however with different metamodels and focusing on refinement, have been studied in [4]. A general context of descriptive transformations employing UML and OCL is nicely described in [5]. The same example with focus on different transformation properties (as consistency and metamodel property preservation) and discussing solving and translation times has been studied in [7], but the covering and completion techniques developed here are not treated there.

The rest of this paper is structured as follows. Section 2 describes the running example. Section 3 sketches how to apply the model validator. Section 4 shows how transformation models can be statically inspected. Section 5 applies our technique for dynamically completing partial transformation model pairs. Section 6 closes the paper with a conclusion and future work.

2 Model Transformation Example

The running example in this paper is the well-known transformation between ER and relational database schemata. We study this transformation in form



```
All association end names are identical to the class name
with lower first letter but without the 'package' name:
trans, erSchema, reDBSchema, entity, relend, relship,
relSchema, attribute and dataType.

The two operations key() and osRelend() are defined as:
class ErSyn_Entity < Base_Named
operations
  key() : Set(Base_Attribute) = self.attribute->select(a | a.isKey)
  osRelend() : Set(ErSyn_Relend) = -- other side relends
    self.relend->collect(re | re.relship.relend->excluding(re))->
      flatten->asSet
end
```

Invariant	Result
Base_Attribute::linkedToOneOfEntityRelshipRelSchema	true
Base_DataType::uniqueDataTypeNames	true
Er2Rel_Trans::forEntityExistsOneRelSchema	true
Er2Rel_Trans::forRelSchemaExistsOneEntityXorRelship	true
Er2Rel_Trans::forRelshipExistsOneRelSchema	true
ErSyn_Entity::differentOsRelendAndAttributeNamesWithinEntity	true
ErSyn_Entity::entityKeyNotEmpty	true
ErSyn_Entity::uniqueAttributeNamesWithinEntity	true
ErSyn_Entity::uniqueOsRelendNamesWithinEntity	true
ErSyn_ErSchema::differentEntityAndRelshipNamesWithinErSchema	true
ErSyn_ErSchema::uniqueEntityNamesWithinErSchema	true
ErSyn_ErSchema::uniqueErSchemaNames	true
ErSyn_ErSchema::uniqueRelshipNamesWithinErSchema	true
ErSyn_Relend::c_Relend_Entity_Relship_ErSchema	true
ErSyn_Relship::differentRelendAndAttributeNamesWithinRelship	true
ErSyn_Relship::relshipKeyEmpty	true
ErSyn_Relship::uniqueAttributeNamesWithinRelship	true
ErSyn_Relship::uniqueRelendNamesWithinRelship	true
RelSyn_RelDBSchema::uniqueRelDBSchemaNames	true
RelSyn_RelDBSchema::uniqueRelSchemaNamesWithinRelDBSchema	true
RelSyn_RelSchema::relSchemaKeyNotEmpty	true
RelSyn_RelSchema::uniqueAttributeNamesWithinRelSchema	true

Constraints ok. (0ms) 100%

Fig. 1. Class diagram and invariants for example transformation model.

of a transformation model as introduced in [6, 3]. A transformation model is a descriptive model where the relationship between source and target is purely characterized by the (source,target) model pairs determined by the transformation. A transformation model consists in our approach of a plain UML class diagram with restricting OCL invariants. Typically, there is an anchor class for the source model, an anchor class for the target model, and a connecting class for the transformation. There are OCL invariants for restricting the source metamodel, for the target metamodel, and for the transformation.

In Fig. 1 the class diagram and the invariant names for the example are pictured. All details of the example can be found in [6]. The example transformation model has four parts: a base part with datatypes and attributes for concepts commonly employed in the ER and relational model; a part for ER schemata (`ErSchema`) with the concepts `Entity`, `Relship` (relationship), and `Relend` (relationship end); a part for relational database schemata (`RelDBSchema`) incorporating relational schemata (`RelSchema`); finally, a part for the transformation (`Trans`). [6] discusses also the semantics. Therefore, some classes here are marked in their names as belonging to the syntax (`ErSyn`, `RelSyn`).

We have used the terms source and target, but transformation models are direction-neutral due to the central employment of associations. We will say that we 'transform a source ER schema into a target relational database schema', but formally the class diagram does not indicate any direction. In our view, transformation models can be looked at as a form of bidirectional transformations.

Currently our model validator does not support the computation of strings in a satisfactory way. In particular, we need string computations for relational attributes in connection with ER attribute names and relationship end names. Through this, we can establish a connection between the source and the target model. Thus, in contrast to [6], we model names (for example, of entities or attributes) as integers and have to pose certain restrictions on the use of the underlying integers and strings. We encode ten letters as digits: A$\leftrightarrow$0, B$\leftrightarrow$1, C$\leftrightarrow$2, D$\leftrightarrow$3, E$\leftrightarrow$4, F$\leftrightarrow$5, G$\leftrightarrow$6, H$\leftrightarrow$7, J$\leftrightarrow$8, K$\leftrightarrow$9. Through a derived attribute `nameS`, we are able to represent the 'integer names' formally as string values. For example, we will calculate: `20 = 2*10+0 ≅` '2.concat(0)' `≅ 'C'.concat('A') = 'CA'`. This section followed the ideas we have developed in [7].

3 Applying the USE Model Validator

We explain the application of the USE model validator by showing how the tool has to be configured in order to construct a model transformation between an example ER schema and a corresponding relational database schema. The needed configuration is shown in Fig. 2 and the resulting generated object diagram, which captures both schemata, is pictured in Fig. 3.

In a model validator configuration, the population of (a) *classes*, (b) *associations*, (c) *attributes* and (d) *datatypes* is determined. Classes, attributes and datatypes are displayed in the configuration table in black-on-white, and associations in black-on-light-grey. (a) A *class* needs an integer upper bound for the maximal number of objects in that class, and an optional lower bound may be given. (b) *Associations* may also have a lower and upper bound for the number of links or their population may be left open and be thus determined through the (up-

```
Er2Rel_Trans : 1..1         -- (a)

Er2Rel_OwnershipTransErSchema    : *
Er2Rel_OwnershipTransRelDBSchema : *
```

```
ErSyn_ErSchema : 1..1
ErSyn_Entity   : 1..1
ErSyn_Relship  : 1..1
ErSyn_Relend   : 2..2

ErSyn_OwnershipErSchemaEntity   : *         -- (b)
ErSyn_OwnershipErSchemaRelship  : *
ErSyn_OwnershipEntityAttribute  : 2..2
ErSyn_OwnershipRelshipAttribute : 0..0      -- (b)
ErSyn_OwnershipRelshipRelend    : *
ErSyn_RelendTyping              : *
```

```
RelSyn_RelDBSchema : 1..1
RelSyn_RelSchema   : 2..2         -- (a)

RelSyn_OwnershipRelDBSchemaRelSchema : *
RelSyn_OwnershipRelSchemaAttribute   : 4..4
```

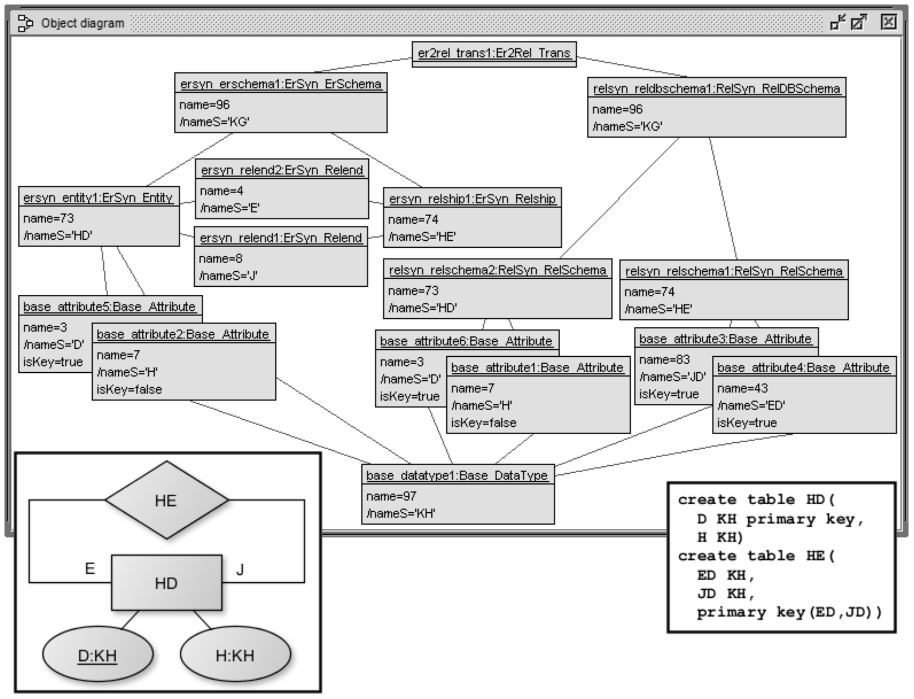
```
Base_Attribute       : 6..6
Base_DataType        : 1..1
Base_Named_name      : Set{0,1,2,3,4,5,6,7,8,9,10, ..., 89,90,91,92,93,94,95,96,97,98,99}
Base_Attribute_isKey : Set{false,true}          -- (c)

Base_AttributeTyping : *

Real      : 0..0
Real_step : 1
String    : 0..0
Integer   : 0..127                              -- (d)
```

```
Class       black-on-white
Association black-on-light-grey
```

Fig. 2. Configuration for ER schema with binary relationship.


Object diagram
er2rel_trans1:Er2Rel_Trans
ersyn_erschema1:ErSyn_ErSchema
name=96
/nameS='KG'
relsyn_reldbschema1:RelSyn_RelDBSchema
name=96
/nameS='KG'
ersyn_relend2:ErSyn_Relend
name=4
/nameS='E'
ersyn_entity1:ErSyn_Entity
name=73
/nameS='HD'
ersyn_relship1:ErSyn_Relship
name=74
/nameS='HE'
ersyn_relend1:ErSyn_Relend
name=8
/nameS='J'
relsyn_relschema2:RelSyn_RelSchema
name=73
/nameS='HD'
relsyn_relschema1:RelSyn_RelSchema
name=74
/nameS='HE'
base_attribute5:Base_Attribute
name=3
/nameS='D'
isKey=true
base_attribute2:Base_Attribute
name=7
/nameS='H'
isKey=false
base_attribute6:Base_Attribute
name=3
/nameS='D'
isKey=true
base_attribute1:Base_Attribute
name=7
/nameS='H'
isKey=false
base_attribute3:Base_Attribute
name=83
/nameS='JD'
isKey=true
base_attribute4:Base_Attribute
name=43
/nameS='ED'
isKey=true
base_datatype1:Base_DataType
name=97
/nameS='KH'
HE
E
J
HD
D:KH
H:KH

```
create table HD(
  D KH primary key,
  H KH)
create table HE(
  ED KH,
  JD KH,
  primary key(ED,JD))
```

Fig. 3. Generated ER and relational database schema with binary relationship.

per bounds for the) participating classes. (c) *Attributes* may be determined by specifying an enumeration of allowed values or by the set of values derived from the value set of the corresponding datatype. (d) The numerical *datatypes* Integer and Real may be configured through an enumeration (e.g., Set{42,44,46} or Set{3.14, 6.28, 9.42}) or with lower and upper bounds for the interval of allowed values with an additional step value for Real (for example, resulting in Set{-8..7} or Set{-1, -0.5, 0, 0.5, 1}). The datatype String may be determined by an enumeration (e.g., Set{'UML','OCL','MDE'}) or through a lower and upper bound for the number of automatically generated String literals (resulting in, for example, Set{'String1',...,'String7'}).

The example configuration requires (among other restrictions) the following: (a) there is exactly one transformation object (in class `Er2Rel_Trans`), and there are exactly two relational schemas (in class `RelSyn_RelSchema`); (b) the links in association `ErSyn_OwnershipErSchemaEntity` between `ErSyn_ErSchema` and `ErSyn_Entity` are not explicitly restricted, but only implicitly through the upper bounds of the participating classes, and there is no link in the association `ErSyn_OwnershipRelshipAttribute`, meaning that in the constructed ER schema there will be no relationship attribute; (c) the attribute isKey is allowed to take values from the enumeration Set{false,true} (recall that in UML and OCL more than two truth values are available); (d) the datatype Integer is allowed to take values from the interval [0..127].

The automatically generated transformation in Fig. 3 is displayed in form of the constructed object diagram and in form of a visual resp. textual domain-specific representation of the ER schema (in traditional ER notation) and the relational database schema (as textual SQL table declarations). In particular, the two relationship ends `E` and `J` of the relationship `HE` are represented as attributes `ED` and `JD` in the relational schema `HE`, because the attribute `D` constitutes the key in entity `HD` and in the relational schema `HD`. If there would be a composed key in the entity `HD`, say attributes `DA` and `DB`, the relational schema `HD` has to contain four attributes `EDA`, `EDB`, `JDA`, and `JDB`. Thus, the key attribute names on the relational side have to be composed from the relationship end and attribute names from the ER side. This section followed the ideas we have developed in [7].

4 Analyzing Static Transformation Model Properties by Coverage of Model Elements

Analyzing static transformation model properties means for us to explore the model transformation text in order to achieve relevant transformation properties. Static analysis is interesting because transformation models are usually structured at least into three parts: (a) the source, (b) the target, and (c) the transformation metamodel. Accompanying constraints will be found in the respective parts. For a single constraint it is thus particular interesting whether it restricts all three parts in conjunction or it treats a single part only. Such an analysis is possible with the static technique proposed here that is based on the idea of covering, i.e., to analyze and to indicate which part of the underlying class diagram is covered by a particular constraint. In our example we even have four parts in the transformation model, namely the source, the target, the transformation, and a common part that represents model features that are used in both the source and the target (here, attributes and datatypes). This is probably not an unusual situation.

In Fig. 4 we have displayed four (of the 22) invariants in the transformation model. Below we show the invariants also in detail. The coloring in the figure indicates the degree the respective model element (here classes, attributes, operations) is used in and covered by the constraint. By inspecting the invariant's color coverage profile one can analyze its effect on the respective model element, and one gets an impression about its dominance.

```
context self:ErSyn_Relend inv c_Relend_Entity_Relship_ErSchema:
  self.entity.erSchema=self.relship.erSchema
context self:ErSyn_Entity inv uniqueOsRelendNamesWithinEntity:
  self.osRelend()->forAll(re1,re2 | re1.name=re2.name implies re1=re2)
context self:Base_Attribute inv linkedToOneOfEntityRelshipRelSchema:
  (self.entity->size)+(self.relship->size)+(self.relSchema->size)=1
context self:Er2Rel_Trans inv forEntityExistsOneRelSchema:
  self.erSchema.entity->forAll(e |
    self.relDBSchema.relSchema->one(rl |
      e.name=rl.name and
      e.attribute->forAll(ea |
        rl.attribute->one(ra |
          ea.name=ra.name and ea.dataType=ra.dataType and
          ea.isKey=ra.isKey))))
```

`ErSyn_Relend::c_Relend_Entity_Relship_ErSchema` basically expresses that the path from `Relend` over `Entity` to `ErSchema` coincides with the path from `Relend` over `Relship` to `ErSchema` (`c_` stands for 'commutativity constraint'). This is reflected in the invariant's coverage profile.

`ErSyn_Entity::uniqueOsRelendNamesWithinEntity` restricts the other-side-relends (`osRelends()`) of an entity. For example, if we have `Person-employee-Job-employer-Company`, then `employer` is an `osRelend` of `Person` and `employee` is an `osRelend` of `Company`. An `osRelend` can be applied to an entity just like an attribute is applied. The collection of the `osRelends` must be unique for an entity. The coverage profile of the constraint clearly expresses that the constraint is working on the ER side only and points out the influence of the operation `osRelend()`.

`Base_Attribute::linkedToOneOfEntityRelshipRelSchema` requires that an `Attribute` either belongs to an `Entity` or to a `Relship` or to a `RelSchema`, but not to more than one model element although the multiplicities would allow this. The coverage profile points to and emphasizes the connection between the four mentioned metaclasses.

`Er2Rel_Trans::forEntityExistsOneRelSchema` is a transformation (`Trans`) constraint and thus covers a large portion of the metamodel, the source, the

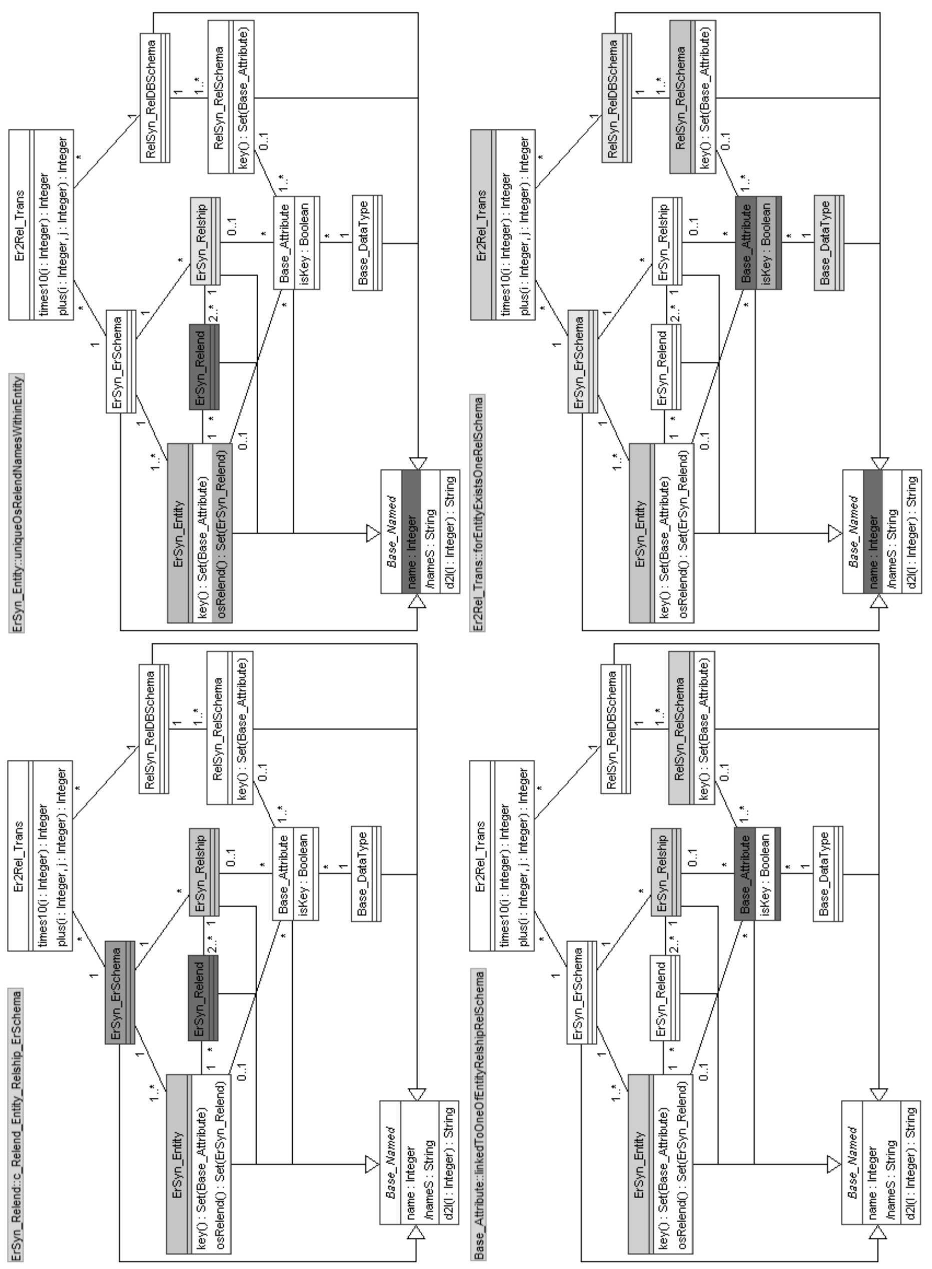

ErSyn_Entity::uniqueOsRelendNamesWithinEntity
Er2Rel_Trans::forEntityExistsOneRelSchema
ErSyn_Relend::c_Relend_Entity_Relship_ErSchema
Base_Attribute::linkedToOneOfEntityRelshipRelSchema
Er2Rel_Trans
times10(i : Integer) : Integer
plus(i : Integer, j : Integer) : Integer
RelSyn_RelDBSchema
RelSyn_RelSchema
key() : Set(Base_Attribute)
ErSyn_ErSchema
ErSyn_Relship
ErSyn_Relend
ErSyn_Entity
key() : Set(Base_Attribute)
osRelend() : Set(ErSyn_Relend)
Base_Attribute
isKey : Boolean
Base_DataType
Base_Named
name : Integer
/nameS : String
d2I(i : Integer) : String

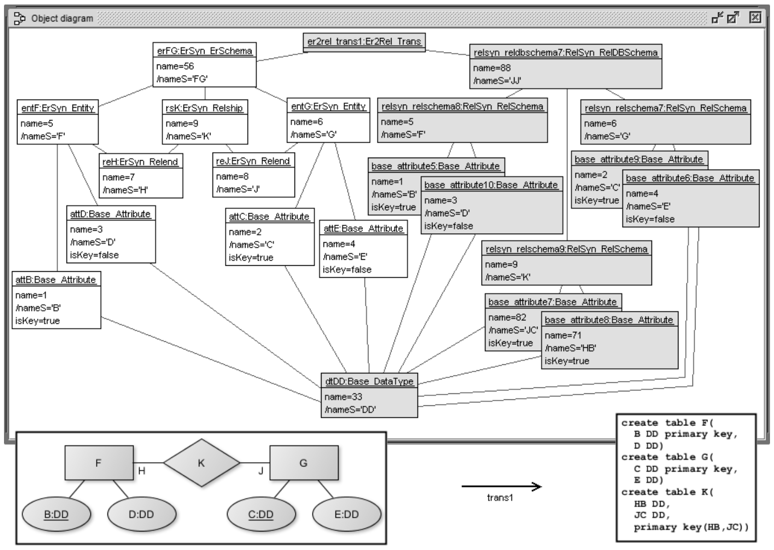
Fig. 4. Coverage of model elements for selected invariants.

target, and the transformation itself. As the constraint deals with `Entity` objects only, the coverage reveals that `Relship` or `Relend` objects are *not* touched. If one goes through all `Trans` constraints, one basically discovers that the operation `osRelend()` is not used in the `Trans` part at all. Thus the coverage indicates that this operation is *not* relevant for the transformation.

Currently we have realized the coverage analysis in the graphical user interface and with predefined metrics. We are considering to represent the analysis results in textual and table form as well and to offer apart from predefined metrics the option to let the developer define her project specific metrics, if desired.

5 Analyzing Dynamic Transformation Model Properties by Transformation Completion

Analyzing dynamic transformation model properties means for us to actually construct transformation instances in form of object diagrams. Doing so can reveal relevant transformation properties. The properties and questions that we



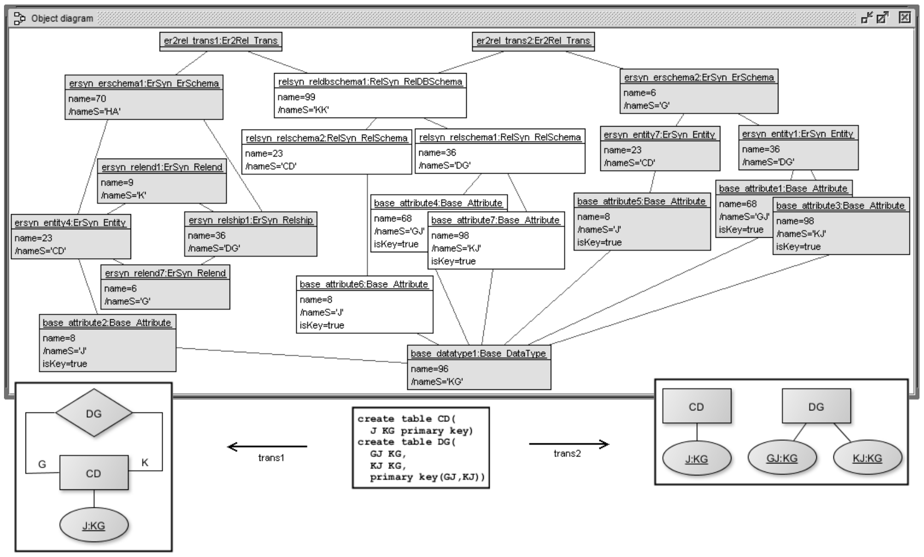
```
create table F(
  B DD primary key,
  D DD)
create table G(
  C DD primary key,
  E DD)
create table K(
  HB DD,
  JC DD,
  primary key(HB,JC))
```

Fig. 5. Transformation completion starting from an ER schema.

consider here are: (a) given a concrete ER schema that is manually constructed, is it possible to automatically complete the transformation yielding a relational database schema and to show by this the effectiveness of the transformation and inspect whether the transformation model constructs the expected result (see Fig. 5) and (b) given a manually constructed relational database schema, is it possible to complete the partially given object diagram and to show that the transformation is non-unique in the sense that the given relational database schema has two ER counterparts (see Fig. 6).

In Fig. 5 question (a) is treated. A partial object diagram representing the manually constructed ER schema (the white objects in the left part of the figure) is handed to the model validator in order to complete the object diagram. The completion is shown in the right part of the figure. The model validator configuration asks for one transformation object, one connected ER schema, and one connected relational database schema.

In Fig. 6 question (b) is handled. A partial object diagram representing the manually constructed relational database schema (the white objects in the middle of the figure) is handed to the model validator in order to complete the object dia-


Object diagram
er2rel_trans1:Er2Rel_Trans
er2rel_trans2:Er2Rel_Trans
ersyn_erschema1:ErSyn_ErSchema name=70 /nameS='HA'
relsyn_reldbschema1:RelSyn_RelDBSchema name=99 /nameS='KK'
ersyn_erschema2:ErSyn_ErSchema name=6 /nameS='G'
relsyn_relschema2:RelSyn_RelSchema name=23 /nameS='CD'
relsyn_relschema1:RelSyn_RelSchema name=36 /nameS='DG'
ersyn_entity7:ErSyn_Entity name=23 /nameS='CD'
ersyn_entity1:ErSyn_Entity name=36 /nameS='DG'
ersyn_relend1:ErSyn_Relend name=9 /nameS='K'
base_attribute4:Base_Attribute name=68 /nameS='GJ' isKey=true
base_attribute7:Base_Attribute name=98 /nameS='KJ' isKey=true
base_attribute5:Base_Attribute name=8 /nameS='J' isKey=true
base_attribute1:Base_Attribute name=68 /nameS='GJ' isKey=true
base_attribute3:Base_Attribute name=98 /nameS='KJ' isKey=true
ersyn_entity4:ErSyn_Entity name=23 /nameS='CD'
ersyn_relship1:ErSyn_Relship name=36 /nameS='DG'
ersyn_relend7:ErSyn_Relend name=6 /nameS='G'
base_attribute6:Base_Attribute name=8 /nameS='J' isKey=true
base_attribute2:Base_Attribute name=8 /nameS='J' isKey=true
base_datatype1:Base_DataType name=96 /nameS='KG'
DG
G
K
CD
J:KG
trans1
trans2
CD
DG
J:KG
GJ:KG
KJ:KG

```
create table CD(
  J KG primary key)
create table DG(
  GJ KG,
  KJ KG,
  primary key(GJ,KJ))
```

Fig. 6. Non-unique transformation completion starting from a relational DB schema.

gram. The ER completions are shown in the left and the right part of the figure. The model validator configuration in this case explicitly asks for two transformation objects connected to a single relational database schema and connected to two different ER schemas. Additionally, two invariants had to be added for the process of finding the proper object diagram. These invariants are not part of the transformation model, but are needed to drive the model validator into the proper direction.

```
context ErSyn_ErSchema inv connectedToTransformation:
  self.trans->notEmpty()
context ErSyn_ErSchema inv oneWithRelship_oneWithoutRelship:
  ErSyn_ErSchema.allInstances()->exists(with,without|
    with.relship->notEmpty() and without.relship->isEmpty())
```

Summarizing, we observe that the approach allows the developer to check the injectivity of a transformation model in either direction. We have considered a transformation model in one particular direction, from the relational database model to the ER model, and were able to show through the construction of an example, that the particular considered direction of the transformation is not injective because one relational database schema was connected with two ER schemas. As the approach is grounded on finite checks, it is not possible to prove in general that a transformation going into one direction is injective, but one can show through examples the non-injectivity.

6 Conclusion

The paper presented an approach for automatically checking transformation model features. We have analyzed transformation models statically by identifying model elements in the underlying class diagram that are covered by a transformation model invariant. We also checked transformation models dynamically through the completion of partially specified transformation pairs.

Future work could consider to study invariant independence, i.e., minimality of transformation models. The static analysis features can be improved by presenting the results in table and text form and through the introduction of project specific definition of metrics. The handling of strings must be improved. Last but not least, larger case studies must check the practicability of the approach.

References

1. Amrani, M., Lucio, L., Selim, G.M.K., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y.L., Cordy, J.R.: A Tridimensional Approach for Studying the Formal Verification of Model Transformations. In Antoniol, G., Bertolino, A., Labiche, Y., eds.: Proc. Workshops ICST, IEEE (2012) 921–928
2. Baudry, B., Ghosh, S., Fleurey, F., France, R.B., Traon, Y.L., Mottu, J.M.: Barriers to Systematic Model Transformation Testing. CACM **53**(6) (2010) 139–143
3. Bezivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model Transformations? Transformation Models! In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Proc. 9th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2006), Springer, Berlin, LNCS 4199 (2006) 440–453
4. Büttner, F., Egea, M., Guerra, E., de Lara, J.: Checking Model Transformation Refinement. In Duddy, K., Kappel, G., eds.: Proc. Inf. Conf. ICMT. LNCS 7909, Springer (2013) 158–173
5. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Verification and Validation of Declarative Model-To-Model Transformations through Invariants. Journal of Systems and Software **83**(2) (2010) 283–302
6. Gogolla, M.: Tales of ER and RE Syntax and Semantics. In Cordy, J.R., Lämmel, R., Winter, A., eds.: Transformation Techniques in Software Engineering, IBFI, Schloss Dagstuhl, Germany (2005) Dagstuhl Seminar Proceedings 05161. 51 pages.
7. Gogolla, M., Hamann, L., Hilken, F.: Checking Transformation Model Properties with a UML and OCL Model Validator. In: Proc. 3rd Int. STAF'2014 Workshop Verification of Model Transformations (VOLT'2014). (2014) CEUR Proceedings.
8. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
9. Kuhlmann, M., Gogolla, M.: From UML and OCL to Relational Logic and Back. In France, R., Kazmeier, J., Breu, R., Atkinson, C., eds.: Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012), Springer, Berlin, LNCS 7590 (2012) 415–431
10. Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2007). (2007) LNCS 4424, 632–647