Checking Transformation Model Properties with a UML and OCL Model Validator

Martin Gogolla, Lars Hamann, Frank Hilken

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

Abstract. This paper studies model transformations in the form of transformation models connecting source and target metamodels. We propose to analyze transformation models with a UML and OCL tool on the basis of an implementation of relational logic on top of Kodkod. Within this approach it is feasible to check for transformation model consistency in different flavors. Certain properties implied by the transformation model, e.g. whether a particular property is preserved by the transformation, can be inspected as well. As an example, the paper uses the well-known transformation between ER schemata and relational database schemata.

1 Introduction

Model transformations are a central ingredient in Model-Driven Engineering (MDE). As for any kind of software artifact, quality improvement techniques like validation and verification of properties are essential for the success of MDE. Verification for model transformation [1] is thus gaining more attention.

This paper discusses model transformations in form of transformation models [2] that are are descriptive and direction-neutral descriptions of mappings between a source and target metamodel. It proposes to check model transformation consistency and implied properties by applying a so-called model validator that searches for model instances within a finite search space of possible instances.

Our work has links to related approaches. Different notions of consistency and instanciability as considered here have also been proposed in [4]. Our work is based on Alloy [7] and its API Kodkod [9]. 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 [8]. Application of transformation models using the same example as employed here, however with different underlying metamodels and focusing on transformation refinement, has been studied in [3]. The application of Alloy in the context of model transformation realized by graph transformations has been recently proposed in [10]. Additional material (complete sources and configurations, and additional examples) can be found in [6].

The rest of this paper is structured as follows. Section 2 describes the running example which is used throughout the paper. Section 3 sketches how to apply

the model validator in the context of example. Section 4 shows how transformation models can be inspected with regard to consistency. Section 5 applies our technique for checking transformation model consequences. Section 6 shortly discusses translation and solving times. Section 7 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 of a transformation model as introduced in [5, 2]. A transformation model is a

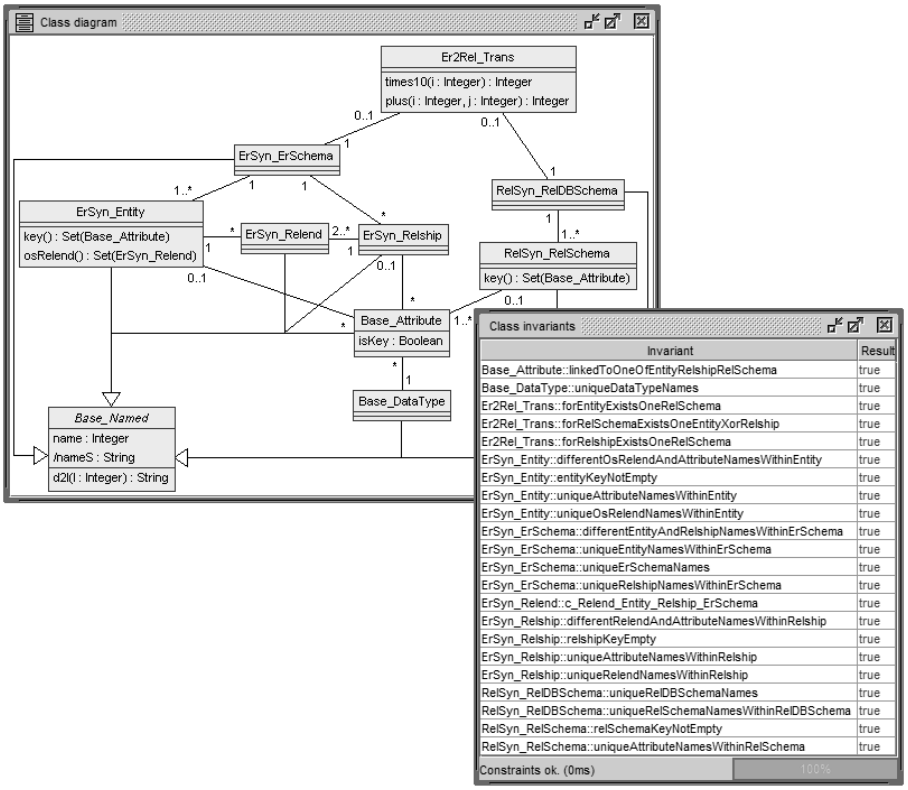


Fig. 1. Class diagram and invariants for example transformation model.

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 [5]. 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`, `Relationship`, and `Relend` (relationship end); a part for relational database schemata (`RelDBSchema`) incorporating relational schemata (`RelSchema`); finally, a part for the transformation (`Trans`). [5] 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 attribute 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 [5], 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 string. However, 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` $\widehat{=}$ '2.concat(0)' $\widehat{=}$ `'C'.concat('A') = 'CA'`.

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 (upper 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* In-

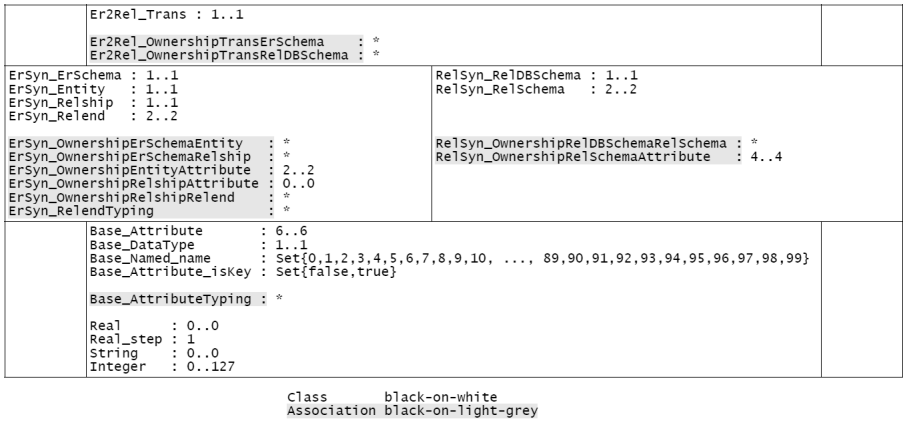


Fig. 2. Configuration for ER schema with binary relationship.

teger 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) resp. 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.

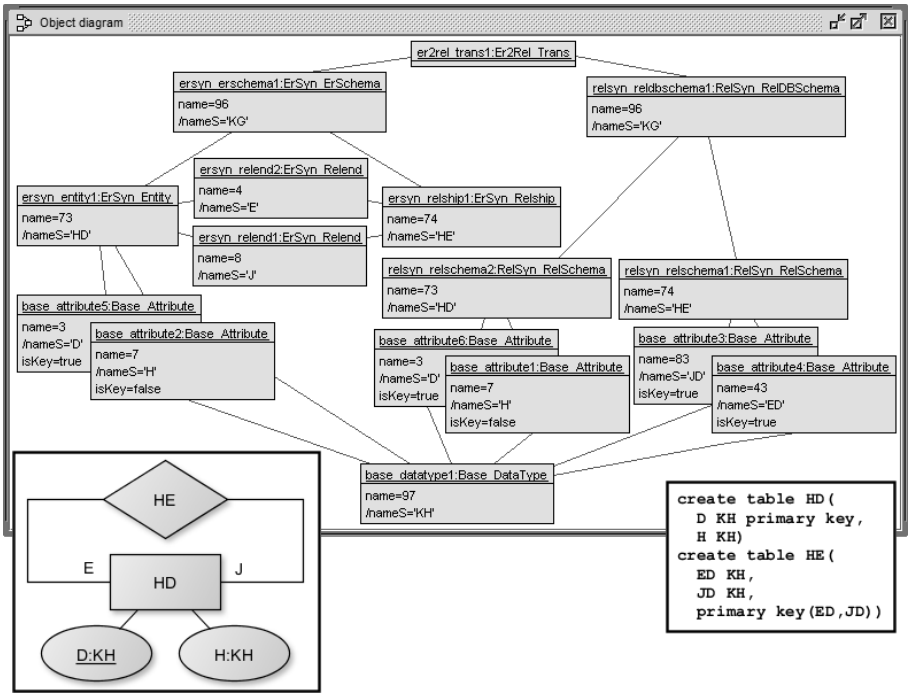


Fig. 3. Generated ER and relational database schema with binary relationship.

4 Checking Transformation Model Consistency

The configuration table in Fig. 4 shows three configurations needed to check for (1) weak consistency, (2) class instanciability, and (3) class and association instanciability. With option (1) we mean that at least one valid object diagram can be found, even with no objects at all or empty populations for a single class, provided the model and the configuration allows this; option (2) means all classes are instantiated with non-empty populations; option (3) means that all classes and all associations are instantiated. The three options are determined by the first, second, and third column of the parts displayed in Fig. 4 with white-on-dark-grey. The main differences between the three options are displayed in the following table.

	weak consistency	class instanciability	class and association instanciability
Class	0..UpperBound	1..UpperBound	1..UpperBound
Association	*	*	1..UpperBound

The first option states an upper bound for class population, but no lower bound for class population, and no restriction for association population. The second option additionally requires all classes to be populated by at least one object.

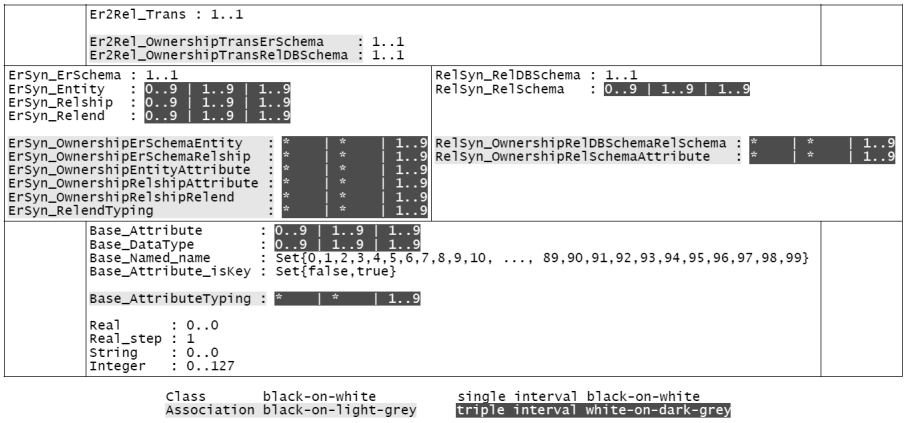


Fig. 4. Configurations for transformation model consistency and instanciability.

The third option additionally demands that all associations must be populated by at least one link. As UpperBound we have chosen in the example the integer 9 which results in manageable execution times. Depending on the considered transformation model, this number may be different. However, the table shows the principle approach to handle consistency and (the two kinds of instanciability) instanciability in our context.

The object diagrams together with the ER schemata and the SQL table definitions in Figs. 5, 6, and 7 show the results of applying the USE model validator in the example transformation model for proving (by example) weak consistency, class instanciability, and class and association instanciability. As the requirements increase, more and more concepts from the metamodel have to be employed by the model validator. In Fig. 5 only entities and attributes are generated on the ER side (indeed only one entity and one attribute), in Fig. 6 additionally relationships and relationship ends show up, and finally in Fig. 7 also relationship attributes occur. The 'Object count' window and the 'Link count' window in Fig. 5 are the means in the USE tool to shortly check the class and association population in an overview. In the weak consistency case there are no relationships and no relationship end objects, and accordingly there are no links for relationships. For the case described in Fig. 6 (class instanciability), the 'Link count' window would show (if displayed) zero links for the association `ErSyn_OwnershipRelshipAttribute` indicating that there are no relationship attributes.

5 Checking Implications of the Transformation Model

Below an additional ad-hoc invariant is specified. The invariant expresses a connection between the key attributes on the ER and the relational side. This invariant is added to the model invariants, and the model validator is started with the weak consistency configuration from Fig. 4 which is the least restrictive configuration (potentially allowing the maximum set of object diagrams).

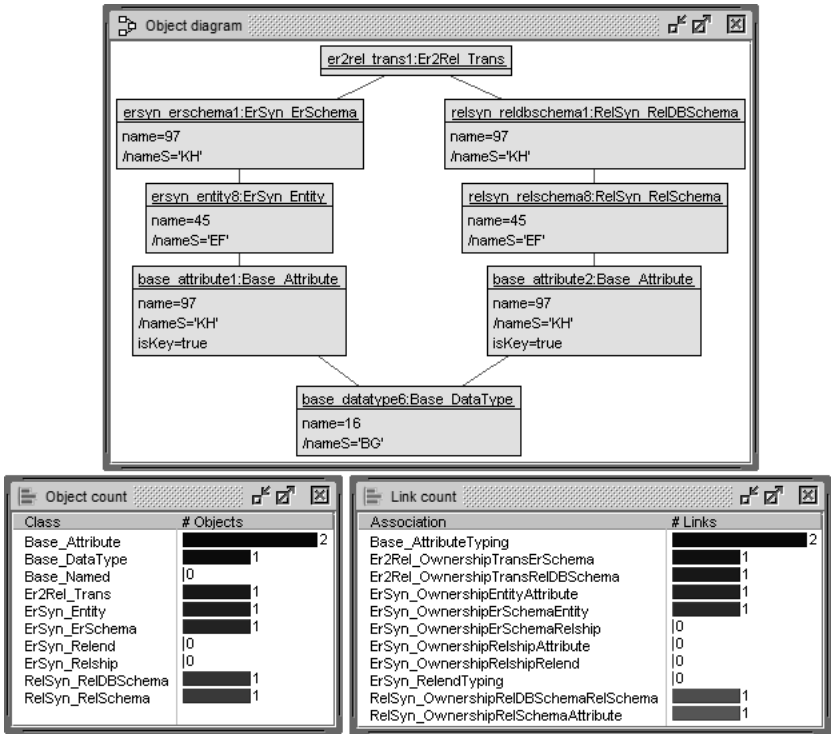


Fig. 5. Generated proof example for weak consistency.

The model validator reports that the model with the additional invariant is not satisfiable.

```
context self:Er2Rel_Trans
  inv erHasOnlyKeyAttrs_relHasSomeNonKeyAttrs:
  self.erSchema.entity.attribute->
    union(self.erSchema.relship.attribute)->
      select(a|a.isKey=false)->isEmpty() and
  self.relDBSchema.relSchema.attribute->
      select(a|a.isKey=false)->notEmpty()

unsatisfiable: erSchema.hasOnlyKeyAttrs() and
                 not relDBSchema.hasOnlyKeyAttrs()

valid:         erSchema.hasOnlyKeyAttrs() implies
                 relDBSchema.hasOnlyKeyAttrs()
```

Of course, the model validator has checked only a finite number of object diagram candidates which are determined by the stated upper bounds. If we conclude from this fact that the model together with the additional invariant is generally

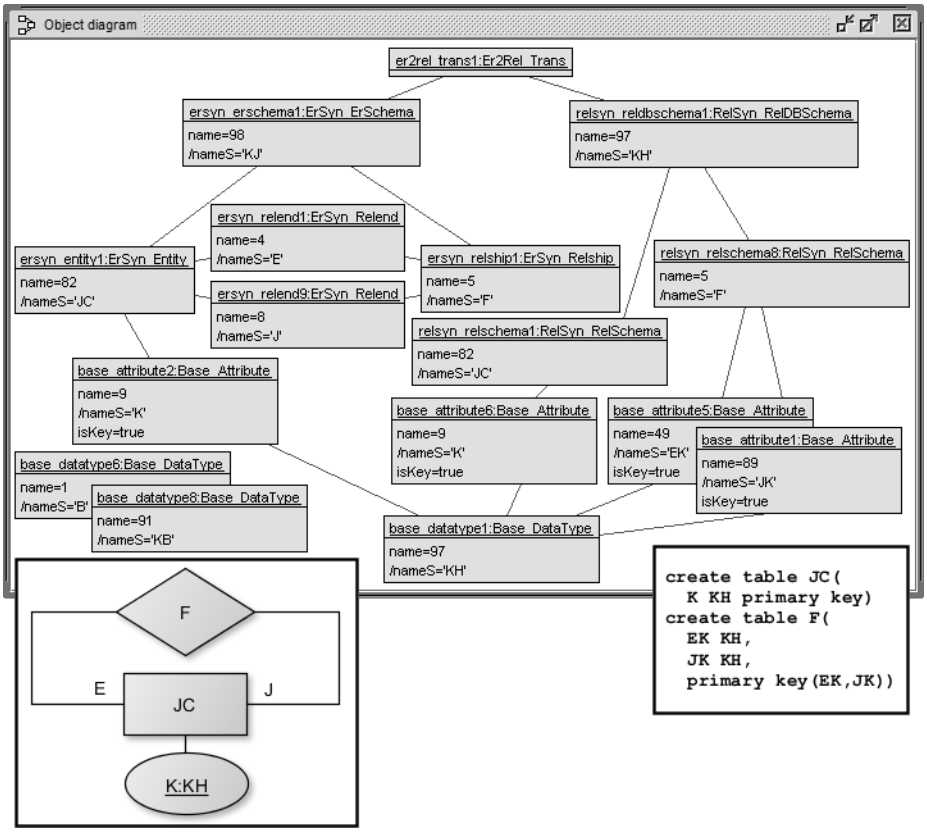


Fig. 6. Generated proof example for class instanciability.

unsatisfiable, then the negated invariant must be valid for the transformation model. We argue that the unsatisfiability ends up in showing that the following property can be formally deduced for the transformation model: if there are only key attributes on the ER side, then there are also only key attributes on the relational side, i.e. non-key attributes cannot exist on the relational side. This is an example where we can show that a property, which can be formulated on the source and target side of the transformation, is preserved when considering the direction from the source to the target.

6 Translation and Solving Times

The table below states translation and solving times. The translation time is the time for translating the configuration into a Kodkod relational formula. The solving time is the time the Kodkod solver needs to compute the answer. The most complex task took about 8 minutes on a standard laptop. We used the Kodkod default solver, but faster solvers are available. Please be aware of the fact that the underlying transformation model has about

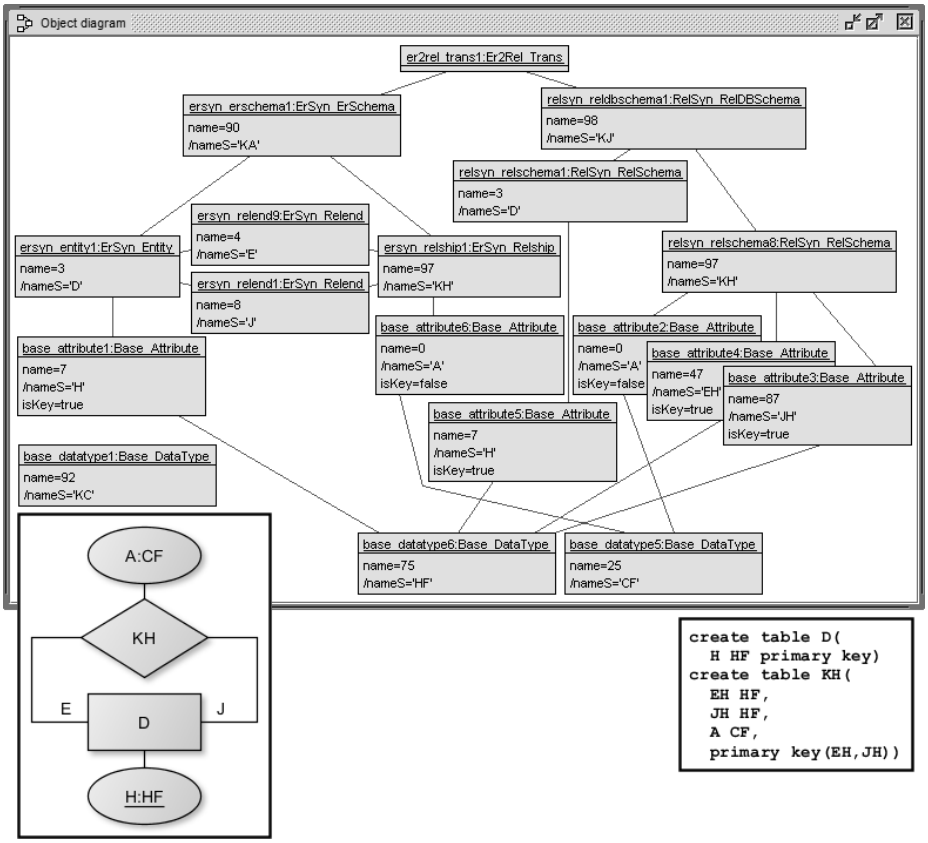


Fig. 7. Generated proof example for class and association instanciability.

20, partly non-trivial OCL constraints. The most complex constraint (see `forRelSchemaExistsOneEntityXorRelship` in [6, page 8]) requires that for every attribute in the relational database schema there either has to be an equivalent entity attribute or a relationship attribute in the ER schema. This constraint is about 20 lines long and has five nested quantifiers.

	Translation [ms]	Solving [ms]
ER schema with binary association	2.152	2.481
Weak consistency	187.002	2.355
Class instanciability	208.258	177.388
Class and association instanciability	184.345	320.705
Implied property ('ER keys' vs. 'rel. keys')	182.957	58.305
Larger example from [6]	12.714	374.650

7 Conclusion

We have presented an approach for automatically checking transformation model features: consistency, class instanciability, class and association instanciability,

and property preservation by the transformation model. The implementation of our model validator is based on the relational model finder Kodkod.

Future work could consider to study invariant independence, i.e., minimality of transformation models. One can also deliver to the model validator a partial solution (for example, a relational database schema) and let the validator automatically complete the transformation (for example, to compute the ER schema). This works in either direction of the transformation. Furthermore, checking for unique results (constructing further results beyond first found solutions) is feasible. 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. 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
3. 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
4. Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL Class Diagrams using Constraint Programming. In: ICST Workshops, IEEE Computer Society (2008) 73–80
5. 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.
6. Gogolla, M., Hamann, L., Hilken, F.: Checking Transformation Model Properties with a UML and OCL Model Validator: Additional Material. Technical report, University of Bremen (2014) `http://www.db.informatik.uni-bremen.de/publications/intern/GHH2014addon.pdf`.
7. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
8. 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
9. 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
10. Wang, X., Büttner, F., Lamo, Y.: Verification of Graph-based Model Transformations Using Alloy. In Hermann, F., Sauer, S., eds.: Proc. Workshop Graph Transformation and Visual Modeling Techniques (GTVMT'2014), ECEASST, Electronic Communications, `journal.ub.tu-berlin.de/eceasst/`. To appear. (2014)

Checking Transformation Model Properties with a UML and OCL Model Validator: Additional Material

Martin Gogolla, Lars Hamann, Frank Hilken

University of Bremen

2. Model Transformation Example: USE File

```
------------------------------------------------------------------------

model er2mof

------------------------------------------------------------------------
--        Er2Rel
--       /    \
--    ErSyn  RelSyn
--       \    /
--        Base
------------------------------------------------------------------------

------------------------------------------------------------------------
-- Base
------------------------------------------------------------------------

abstract class Base_Named
attributes
  -- assume for attribute name: 0 <= name <= 99
  name : Integer -- String
  nameS : String derived =
    let l1=(name div 10) in let l2=(name.mod(10)) in
    if name<10 then d2l(l2) else d2l(l1).concat(d2l(l2)) endif
operations
  d2l(l:Integer):String=
    if l=0 then 'A' else
    if l=1 then 'B' else
    if l=2 then 'C' else
    if l=3 then 'D' else
    if l=4 then 'E' else
    if l=5 then 'F' else
    if l=6 then 'G' else
    if l=7 then 'H' else
    if l=8 then 'J' else
    if l=9 then 'K' else null
    endif endif endif endif endif endif endif endif endif endif
end

class Base_DataType < Base_Named
end

class Base_Attribute < Base_Named
attributes
  isKey : Boolean;
end
```

```
------------------------------------------------------------------------
-- ErSyn
------------------------------------------------------------------------

class ErSyn_ErSchema < Base_Named
end

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

class ErSyn_Relship < Base_Named
end

class ErSyn_Relend < Base_Named
end

------------------------------------------------------------------------
-- RelSyn
------------------------------------------------------------------------

class RelSyn_RelDBSchema < Base_Named
end

class RelSyn_RelSchema < Base_Named
operations
  key() : Set(Base_Attribute) = self.attribute->select(a | a.isKey)
end

------------------------------------------------------------------------
-- Er2Rel
------------------------------------------------------------------------

class Er2Rel_Trans
operations
  times10(i:Integer):Integer=
    if i=0 then  0 else
    if i=1 then 10 else
    if i=2 then 20 else
    if i=3 then 30 else
    if i=4 then 40 else
    if i=5 then 50 else
    if i=6 then 60 else
    if i=7 then 70 else
    if i=8 then 80 else
    if i=9 then 90 else null
    endif endif endif endif endif endif endif endif endif endif
```

```
  plus(i:Integer,j:Integer):Integer=
    if j=0 then i   else
    if j=1 then i+1 else
    if j=2 then i+2 else
    if j=3 then i+3 else
    if j=4 then i+4 else
    if j=5 then i+5 else
    if j=6 then i+6 else
    if j=7 then i+7 else
    if j=8 then i+8 else
    if j=9 then i+9 else null
    endif endif endif endif endif endif endif endif endif endif
end

-------------------------------------------------------------------------
-- Base
-------------------------------------------------------------------------

association Base_AttributeTyping between
  Base_Attribute[0..*] role attribute;
  Base_DataType[1] role dataType;
end

-------------------------------------------------------------------------
-- ErSyn
-------------------------------------------------------------------------

-- ownerships assocs

association ErSyn_OwnershipErSchemaEntity between
  ErSyn_ErSchema[1] role erSchema;
  ErSyn_Entity[1..*] role entity;
end

association ErSyn_OwnershipErSchemaRelship between
  ErSyn_ErSchema[1] role erSchema;
  ErSyn_Relship[0..*] role relship;
end

association ErSyn_OwnershipRelshipRelend between
  ErSyn_Relship[1] role relship;
  ErSyn_Relend[2..*] role relend;
end

association ErSyn_OwnershipEntityAttribute between
  ErSyn_Entity[0..1] role entity;
  Base_Attribute[0..*] role attribute;
end

association ErSyn_OwnershipRelshipAttribute between
  ErSyn_Relship[0..1] role relship;
  Base_Attribute[0..*] role attribute;
end
```

```
-- Typing associations

association ErSyn_RelendTyping between
  ErSyn_Relend[0..*] role relend;
  ErSyn_Entity[1] role entity;
end

--------------------------------------------------------------------------
-- RelSyn
--------------------------------------------------------------------------

association RelSyn_OwnershipRelDBSchemaRelSchema between
  RelSyn_RelDBSchema[1] role relDBSchema;
  RelSyn_RelSchema[1..*] role relSchema;
end

association RelSyn_OwnershipRelSchemaAttribute between
  RelSyn_RelSchema[0..1] role relSchema;
  Base_Attribute[1..*] role attribute;
end

--------------------------------------------------------------------------
-- Er2Rel
--------------------------------------------------------------------------

association Er2Rel_OwnershipTransErSchema between
  Er2Rel_Trans[0..1] role trans;
  ErSyn_ErSchema[1] role erSchema;
end

association Er2Rel_OwnershipTransRelDBSchema between
  Er2Rel_Trans[0..1] role trans;
  RelSyn_RelDBSchema[1] role relDBSchema;
end

--------------------------------------------------------------------------

constraints

--------------------------------------------------------------------------
-- Base
--------------------------------------------------------------------------

-- Commutativity restriction: The DataType of the Attribute of an
-- AttrMap is identical to the DataType of the Value of the AttrMap

context self:Base_AttrMap inv c_AttrMap_Attribute_Value_DataType:
  self.attribute.dataType=self.value.dataType */

-- Naming restriction: Different DataTypes have different names

context self:Base_DataType inv uniqueDataTypeNames:
  Base_DataType.allInstances->
    forAll(self2 | self.name=self2.name implies self=self2)
```

```
------------------------------------------------------------------------
-- ErSyn
------------------------------------------------------------------------

-- Commutativity restriction: The ErSchema of the Entity of a Relend
-- is identical to the ErSchema of the Relship of the Relend

context self:ErSyn_Relend inv c_Relend_Entity_Relship_ErSchema:
  self.entity.erSchema=self.relship.erSchema

-- Name restrictions - - - - - - - - - - - - - - - - - - - - - - - - -

-- Different ErSchemas have different names

context self:ErSyn_ErSchema inv uniqueErSchemaNames:
  ErSyn_ErSchema.allInstances->
    forAll(self2 | self.name=self2.name implies self=self2)

-- Within one ErSchema, different Entities have different names

context self:ErSyn_ErSchema inv uniqueEntityNamesWithinErSchema:
  self.entity->forAll(e1,e2 | e1.name=e2.name implies e1=e2)

-- Within one ErSchema, different Relships have different names

context self:ErSyn_ErSchema inv uniqueRelshipNamesWithinErSchema:
  self.relship->forAll(r1,r2 | r1.name=r2.name implies r1=r2)

-- Within one ErSchema, Entities and Relships have different names

context self:ErSyn_ErSchema
  inv differentEntityAndRelshipNamesWithinErSchema:
  self.entity->forAll(e | self.relship->forAll(r | e.name<>r.name))

-- Within one Relship, different Relends have different names

context self:ErSyn_Relship inv uniqueRelendNamesWithinRelship:
  self.relend->forAll(re1,re2 | re1.name=re2.name implies re1=re2)

-- Within one Entity, different Attributes have different names

context self:ErSyn_Entity inv uniqueAttributeNamesWithinEntity:
  self.attribute->forAll(a1,a2 | a1.name=a2.name implies a1=a2)

-- Within one Relship, different Attributes have different names

context self:ErSyn_Relship inv uniqueAttributeNamesWithinRelship:
  self.attribute->forAll(a1,a2 | a1.name=a2.name implies a1=a2)
```

```
-- Within one Entity, opposite side Relends and Attributes have
-- different names

context self:ErSyn_Entity
  inv differentOsRelendAndAttributeNamesWithinEntity:
  self.osRelend()->forAll(re | self.attribute->forAll(a |
    re.name<>a.name))

-- Within one Relship, Relends and Attributes have different names

context self:ErSyn_Relship
  inv differentRelendAndAttributeNamesWithinRelship:
  self.relend->forAll(re | self.attribute->forAll(a | re.name<>a.name))

-- Within one Entity, different opposite side Relends have different
-- names

context self:ErSyn_Entity inv uniqueOsRelendNamesWithinEntity:
  self.osRelend()->forAll(re1,re2 | re1.name=re2.name implies re1=re2)

-- Key restrictions  - - - - - - - - - - - - - - - - - - - - - - - - - -

-- The set of key attributes of an Entity is not empty

context self:ErSyn_Entity inv entityKeyNotEmpty:
  self.key()->notEmpty

-- The set of key attributes of a Relship is empty

context self:ErSyn_Relship inv relshipKeyEmpty:
  self.attribute->select(a | a.isKey)->isEmpty

-------------------------------------------------------------------------
-- RelSyn
-------------------------------------------------------------------------

-- Name restrictions - - - - - - - - - - - - - - - - - - - - - - - - - -

-- Different RelDBSchemas have different names

context self:RelSyn_RelDBSchema inv uniqueRelDBSchemaNames:
  RelSyn_RelDBSchema.allInstances->forAll(self2 |
    self.name=self2.name implies self=self2)

-- Within one RelDBSchema, different RelSchemas have different names

context self:RelSyn_RelDBSchema
  inv uniqueRelSchemaNamesWithinRelDBSchema:
  self.relSchema->forAll(r1,r2 | r1.name=r2.name implies r1=r2)

-- Within one RelSchema, different Attributes have different names

context self:RelSyn_RelSchema inv uniqueAttributeNamesWithinRelSchema:
  self.attribute->forAll(a1,a2 | a1.name=a2.name implies a1=a2)
```

```
-- Key restriction - - - - - - - - - - - - - - - - - - - - - - - - - - - -

-- The set of key Attributes of a RelSchema is not empty

context self:RelSyn_RelSchema inv relSchemaKeyNotEmpty:
  self.key()->notEmpty

-------------------------------------------------------------------------
-- DataMods - Package only with constraints; no classes; no associations
-------------------------------------------------------------------------

-- An Attribute is either an Entity Attribute or a Relship Attribute
-- or a RelSchema Attribute

context self:Base_Attribute inv linkedToOneOfEntityRelshipRelSchema:
  (self.entity->size)+(self.relship->size)+(self.relSchema->size)=1

-------------------------------------------------------------------------
-- Er2Rel
-------------------------------------------------------------------------

-- Constraints on syntax part  - - - - - - - - - - - - - - - - - - - - -

-- For every Entity in the ErSchema there is a RelSchema having the
-- same name and Attributes with the same properties, i.e., name,
-- DataType, and key property

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))))

-- For every Relship in the ErSchema there is a RelSchema having the
-- same name, Relends representing the arms of the relationship, and
-- Attributes with the same properties, i.e., name, DataType, and key
-- property

context self:Er2Rel_Trans inv forRelshipExistsOneRelSchema:
  self.erSchema.relship->forAll(rs |
    self.relDBSchema.relSchema->one(rl |
      rs.name=rl.name and
      rs.relend->forAll(re | re.entity.key()->forAll(rek |
        rl.attribute->one(ra |
          -- re.name.concat('_').concat(rek.name)=ra.name and
          -- re.name*10+rek.name=ra.name and
          plus(times10(re.name),rek.name)=ra.name and
          rek.dataType=ra.dataType and ra.isKey))) and
      rs.attribute->forAll(rsa |
        rl.attribute->one(ra |
          rsa.name=ra.name and rsa.dataType=ra.dataType and
          ra.isKey=false))))
```

```
-- For every RelSchema there is either an Entity or a Relship with the
-- same properties and name; if the RelSchema corresponds to an
-- Entity, both have Attributes with the same names, DataTypes, and
-- key properties; if the RelSchema corresponds to a Relship, the
-- RelSchema has Attributes corresponding to the arms of the Relship
-- and both have Attributes with the same properties

context self:Er2Rel_Trans inv forRelSchemaExistsOneEntityXorRelship:
  self.relDBSchema.relSchema->forAll(rl |
    self.erSchema.entity->one(e |
      rl.name=e.name and
      rl.attribute->forAll(ra |
        e.attribute->one(ea |
          ra.name=ea.name and ea.dataType=ra.dataType and
          ra.isKey=ea.isKey)))
    xor
    self.erSchema.relship->one(rs |
      rl.name=rs.name and
      rl.attribute->forAll(ra |
        rs.relend->one(re |
          re.entity.key()->one(rek |
            -- ra.name=re.name.concat('_').concat(rek.name) and
            -- ra.name=re.name*10+rek.name and
            ra.name=plus(times10(re.name),rek.name) and
            ra.dataType=rek.dataType and ra.isKey))
        xor
        rs.attribute->one(rsa |
          ra.name=rsa.name and ra.dataType=rsa.dataType and
          ra.isKey=false))))

------------------------------------------------------------------------

context self:ErSyn_Relend inv relendNameNotZero:
  1<=self.name

------------------------------------------------------------------------
```

3. Applying the USE Model Validator

Configuration for ER schema with binary relationship

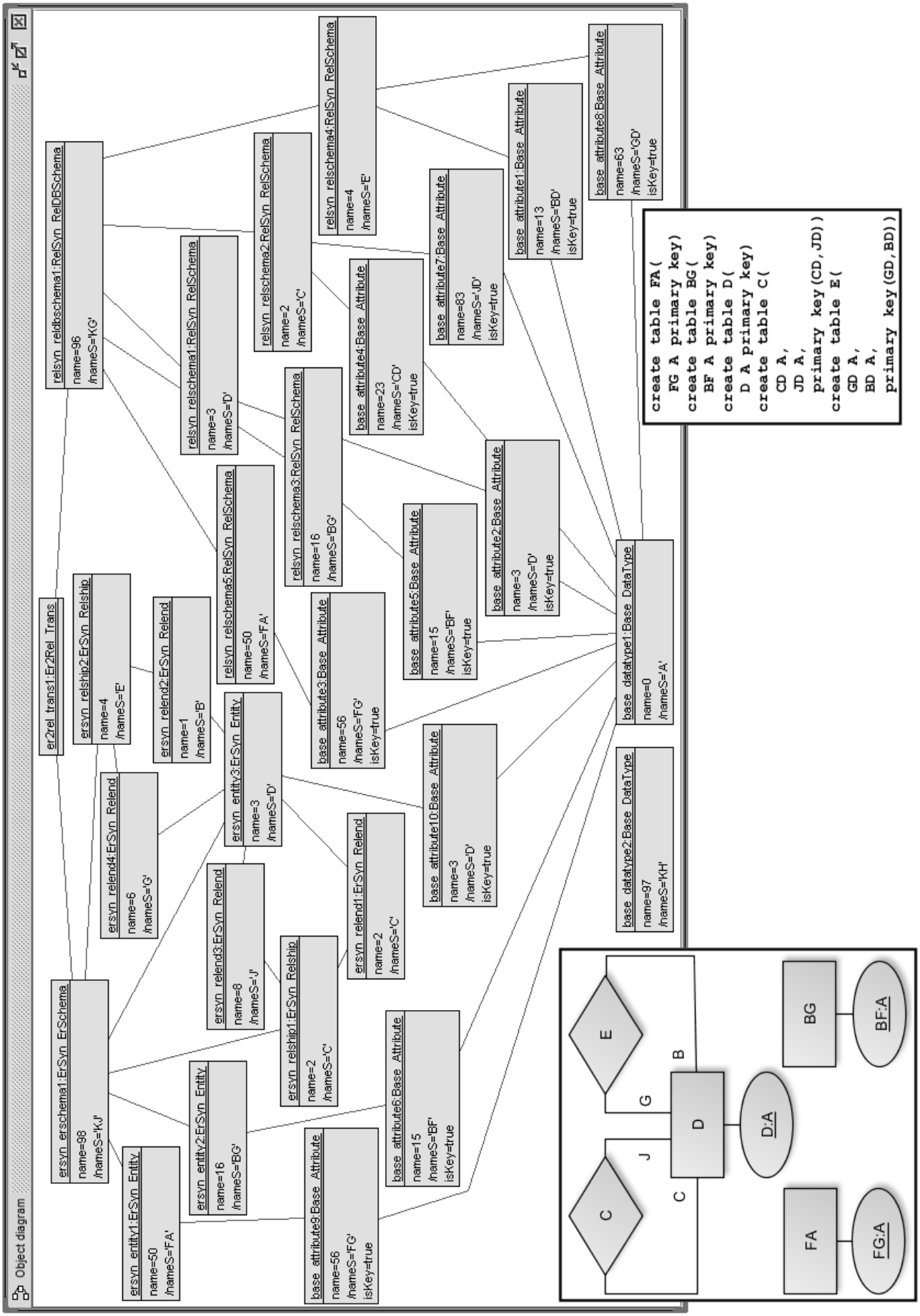
```
Base_Named_name = Set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,2
7,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99}

Base_Attribute_min = 6
Base_Attribute_max = 6

Base_Attribute_isKey = Set{false,true}

Base_DataType_min = 1
Base_DataType_max = 1

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Base_AttributeTyping_min = 0
Base_AttributeTyping_max = -1

------------------------------------------------------------------------

ErSyn_ErSchema_min = 1
ErSyn_ErSchema_max = 1

ErSyn_Entity_min = 1
ErSyn_Entity_max = 1

ErSyn_Relship_min = 1
ErSyn_Relship_max = 1

ErSyn_Relend_min = 2
ErSyn_Relend_max = 2

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ErSyn_OwnershipErSchemaEntity_min = 0
ErSyn_OwnershipErSchemaEntity_max = -1

ErSyn_OwnershipErSchemaRelship_min = 0
ErSyn_OwnershipErSchemaRelship_max = -1

ErSyn_OwnershipEntityAttribute_min = 2
ErSyn_OwnershipEntityAttribute_max = 2

ErSyn_OwnershipRelshipAttribute_min = 0
ErSyn_OwnershipRelshipAttribute_max = 0

ErSyn_OwnershipRelshipRelend_min = 0
ErSyn_OwnershipRelshipRelend_max = -1

ErSyn_RelendTyping_min = 0
ErSyn_RelendTyping_max = -1
```

```
-----------------------------------------------------------------------

RelSyn_RelDBSchema_min = 1
RelSyn_RelDBSchema_max = 1

RelSyn_RelSchema_min = 2
RelSyn_RelSchema_max = 2

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RelSyn_OwnershipRelDBSchemaRelSchema_min = 0
RelSyn_OwnershipRelDBSchemaRelSchema_max = -1

RelSyn_OwnershipRelSchemaAttribute_min = 4
RelSyn_OwnershipRelSchemaAttribute_max = 4

-----------------------------------------------------------------------

Er2Rel_Trans_min = 1
Er2Rel_Trans_max = 1

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Er2Rel_OwnershipTransErSchema_min = 0
Er2Rel_OwnershipTransErSchema_max = -1

Er2Rel_OwnershipTransRelDBSchema_min = 0
Er2Rel_OwnershipTransRelDBSchema_max = -1

-----------------------------------------------------------------------

Real_min = 0
Real_max = 0
Real_step = 1

String_min = 0
String_max = 0

Integer_min = 0
Integer_max = 127
```

```
4. Checking Transformation Model Consistency

Configuration for transformation model consistency

Base_Named_name = Set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,2
7,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99}

Base_Attribute_min = 0
Base_Attribute_max = 9

Base_Attribute_isKey = Set{false,true}

Base_DataType_min = 0
Base_DataType_max = 9

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Base_AttributeTyping_min = 0
Base_AttributeTyping_max = -1

------------------------------------------------------------------------

ErSyn_ErSchema_min = 1
ErSyn_ErSchema_max = 1

ErSyn_Entity_min = 0
ErSyn_Entity_max = 9

ErSyn_Relship_min = 0
ErSyn_Relship_max = 9

ErSyn_Relend_min = 0
ErSyn_Relend_max = 9

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ErSyn_OwnershipErSchemaEntity_min = 0
ErSyn_OwnershipErSchemaEntity_max = -1

ErSyn_OwnershipErSchemaRelship_min = 0
ErSyn_OwnershipErSchemaRelship_max = -1

ErSyn_OwnershipEntityAttribute_min = 0
ErSyn_OwnershipEntityAttribute_max = -1

ErSyn_OwnershipRelshipAttribute_min = 0
ErSyn_OwnershipRelshipAttribute_max = -1

ErSyn_OwnershipRelshipRelend_min = 0
ErSyn_OwnershipRelshipRelend_max = -1

ErSyn_RelendTyping_min = 0
ErSyn_RelendTyping_max = -1
```

```
-----------------------------------------------------------------------

RelSyn_RelDBSchema_min = 1
RelSyn_RelDBSchema_max = 1

RelSyn_RelSchema_min = 0
RelSyn_RelSchema_max = 9

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RelSyn_OwnershipRelDBSchemaRelSchema_min = 0
RelSyn_OwnershipRelDBSchemaRelSchema_max = -1

RelSyn_OwnershipRelSchemaAttribute_min = 0
RelSyn_OwnershipRelSchemaAttribute_max = -1

-----------------------------------------------------------------------

Er2Rel_Trans_min = 1
Er2Rel_Trans_max = 1

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Er2Rel_OwnershipTransErSchema_min = 1
Er2Rel_OwnershipTransErSchema_max = 1

Er2Rel_OwnershipTransRelDBSchema_min = 1
Er2Rel_OwnershipTransRelDBSchema_max = 1

-----------------------------------------------------------------------

Real_min = 0
Real_max = 0
Real_step = 1

String_min = 0
String_max = 0

Integer_min = 0
Integer_max = 127
```

8. Additional Larger Example: Three entities and two relationships



```
Base_Named_name = Set
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,2
7,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51
,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99}

Base_Attribute_min = 10
Base_Attribute_max = 10

Base_Attribute_isKey = Set{true}

Base_DataType_min = 2
Base_DataType_max = 2

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Base_AttributeTyping_min = 10
Base_AttributeTyping_max = 10

------------------------------------------------------------------------

ErSyn_ErSchema_min = 1
ErSyn_ErSchema_max = 1

ErSyn_Entity_min = 3
ErSyn_Entity_max = 3

ErSyn_Relship_min = 2
ErSyn_Relship_max = 2

ErSyn_Relend_min = 4
ErSyn_Relend_max = 4

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ErSyn_OwnershipErSchemaEntity_min = 3
ErSyn_OwnershipErSchemaEntity_max = 3

ErSyn_OwnershipErSchemaRelship_min = 2
ErSyn_OwnershipErSchemaRelship_max = 2

ErSyn_OwnershipEntityAttribute_min = 3
ErSyn_OwnershipEntityAttribute_max = 3

ErSyn_OwnershipRelshipAttribute_min = 0
ErSyn_OwnershipRelshipAttribute_max = 0

ErSyn_OwnershipRelshipRelend_min = 4
ErSyn_OwnershipRelshipRelend_max = 4

ErSyn_RelendTyping_min = 4
ErSyn_RelendTyping_max = 4

------------------------------------------------------------------------

RelSyn_RelDBSchema_min = 1
RelSyn_RelDBSchema_max = 1
```

```
RelSyn_RelSchema_min = 5
RelSyn_RelSchema_max = 5

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

RelSyn_OwnershipRelDBSchemaRelSchema_min = 5
RelSyn_OwnershipRelDBSchemaRelSchema_max = 5

RelSyn_OwnershipRelSchemaAttribute_min = 7
RelSyn_OwnershipRelSchemaAttribute_max = 7

-----------------------------------------------------------------------

Er2Rel_Trans_min = 1
Er2Rel_Trans_max = 1

-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Er2Rel_OwnershipTransErSchema_min = 1
Er2Rel_OwnershipTransErSchema_max = 1

Er2Rel_OwnershipTransRelDBSchema_min = 1
Er2Rel_OwnershipTransRelDBSchema_max = 1

-----------------------------------------------------------------------

Real_min = 0
Real_max = 0
Real_step = 1

String_min = 0
String_max = 0

Integer_min = 0
Integer_max = 127
```